RENATO MANETTA BERNARDINO

# Mapeamento do modelo de objetos para o modelo relacional com a utilização do *Entity Framework* - Estudo de Caso

São Paulo
2011

**RENATO MANETTA BERNARDINO**

# Mapeamento do modelo de objetos para o modelo relacional com a utilização do *Entity Framework* - Estudo de Caso

Monografia apresentada à Escola Politécnica da Universidade de São Paulo, como parte dos requisitos para obtenção do título de MBA em Tecnologia da Informação.

Orientadora:
Dra. Solange Nice Alves Souza

**São Paulo**
**2011**

## FICHA CATALOGRÁFICA



Bernardino, Renato Manetta
Mapeamento do modelo de objetos para o modelo relacional com a utilização do entity framework / R.M. Bernardino. -- São Paulo, 2011.
70 p.

Monografia (MBA em Tecnologia da Informação) - Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1. Frameworks 2. Banco de dados relacionais 3. UML 4. Programação orientada a objetos 5. Desenvolvimento de software 6. Gerenciadores de banco de dados I. Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II. t.

# RESUMO

No desenvolvimento de programas com orientação a objetos, um dos problemas encontrados, é como realizar a persistência dos objetos da aplicação após a sua criação ou processamento em um meio de armazenamento duradouro. Com o objetivo de minimizar os impactos da utilização de Banco de Dados Relacional para a persistência de objetos, nos últimos anos, foram desenvolvidas ferramentas para a automatização do procedimento de mapeamento e abstração do acesso aos dados. Essas ferramentas, chamadas de *framework* de persistência, criam uma interface de programação simples e retiram do desenvolvedor a necessidade de conhecimento aprofundado de Sistema Gerenciador de Banco de Dados (SGBD) e da linguagem de consulta e manipulação de dados para tais ambientes. Nessa pesquisa utiliza-se o *Entity Framework* para auxiliar no processo de mapeamento do modelo de objeto para o relacional. Devido ao estado atual da aplicação usada como caso de estudo nesta pesquisa, o modelo de classe UML foi construído para capturar a estrutura de atributos, operações e relacionamentos existentes entre o conjunto de objetos. Padrões de projeto guiaram a avaliação do modelo de classes e sua adequação para o emprego do *Entity Framework.* Como parte do trabalho, é feita uma avaliação do esquema de banco de dados gerado pela ferramenta, tal avaliação é feita com base no nível de normalização das tabelas geradas pela ferramenta. Ainda como parte do trabalho apresenta-se um guia do emprego do *Entity Framework*, ressaltando-se facilidades, dificuldades, vantagens e desvantagens encontradas.

**Palavras-chave:** *Entity Framework*. UML. Padrões de projeto. Banco de dados. Normalização. Mapeamento objeto-relacional.

# ABSTRACT

In the development of object-oriented programs, one of the problems encountered is how to accomplish the persistence of the application objects after their creation and processing in a durable storage medium. Aiming to minimize the impacts of using a relational database for object persistence in recent years, some tools have been developed to automate the mapping of procedure and abstraction of data access. These tools, called the persistence framework, create a simple programming interface and remove the developer's need for deep knowledge of the Management System Database (DBMS) and the query and data manipulation language for such environments. In this research is used the Entity Framework to assist in the process of mapping the object model to the relational model. Due to the current state of the application used as a case study in this research, the UML class model was built to capture the structure of attributes, operations and existing relationships among the set of objects. Design patterns guided the evaluation of model classes and their suitability for the use of the Entity Framework. An assessment, as part of the work, is made of the database schema generated by the tool, based on the normalization level of the tables generated by the tool. Also as part of the paper, a guide to the use of the Entity Framework is presented, highlighting the facilities, difficulties, advantages and disadvantages encountered.



**Keywords:** Entity Framework. UML. Design Patterns. Database. Standardization. Object-relational mapping.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| **ADVPL** | *Advanced Protheus Language* |
| **CSDL** | *Conceptual Schema Definition Language* |
| **DBA** | *DataBase Administrator* |
| **EDM** | *Entity Data Model* |
| **ERP** | *Enterprise Resource Planning* |
| **IDE** | *Integrated Development Environment* |
| **LINQ** | *Language Integrated Query* |
| **MER** | Modelo Entidade-Relacionamento |
| **MLS** | *Mapping Specification Language* |
| **OID** | *Object Identifier* |
| **POCO** | *Plain Old CLR Object* |
| **POJO** | *Plain Old Java Objects* |
| **SGBD** | Sistema Gerenciador de Banco de Dados |
| **SQL** | *Structured Query Language* |
| **SSDL** | *Store Schema Definition Language* |
| **TI** | Tecnologia da Informação |
| **UML** | *Unified Modeling Language* |
| **XML** | *Extensible Markup Language* |

# SUMÁRIO

# 1 INTRODUÇÃO

No âmbito do desenvolvimento de *software*, referente à persistência do estado dos objetos, em diversos sistemas corporativos ocorre um descasamento de impedâncias entre a tecnologia de orientação a objetos e bancos de dados relacionais. Atualmente, essas tecnologias divergentes coexistem e interferem diretamente na construção dos sistemas, mesclando suas características e proporcionando, em um primeiro momento, redução de custos, manutenção, entre outros. Porém, ao longo dos anos, pode ser notado um aumento do acoplamento entre as camadas distintas das aplicações e a construção de barreiras para uma evolução tecnológica.

## 1.1 MOTIVAÇÃO

Decorrente da mudança constante nos requisitos de negócio da área comercial de uma empresa do ramo têxtil, objeto de estudo e nome fictício Gratex, e com base nos requisitos não-funcionais dos seus sistemas como *e-commerce*[1] e *BackOffice*[2], a equipe do desenvolvimento de *software* utiliza recursos como orientação a objetos e Banco de Dados relacional por encontrar, principalmente, as seguintes vantagens:

a) Tecnologia amadurecida e diversidade de fornecedores;
b) Facilidade para manutenção dos sistemas;
c) Recurso humano capacitado.

Na empresa: a constante necessidade por melhoria nos sistemas, com o objetivo de prover alinhamento entre o negócio e a TI; a compra e troca por um

[1] *E-commerce* (comércio eletrônico em português) é o processo de compra, venda e troca de produtos, serviços e informações por redes de computadores ou pela Internet.

[2] *BackOffice* refere-se ao sistema (*software*) interno que suporta a atividade empresarial.

novo sistema gerenciador de Banco de Dados; e o aprendizado das ferramentas de mapeamento objeto-relacional, *framework* de persistência; evidenciam a motivação dessa pesquisa.

Além desses fatores, a concepção de sistemas complementares utilizando um Banco de Dados relacional compartilhado, proporciona um ambiente diferenciado para a aplicação de *framework* de persistência, onde existe o desafio de definir procedimentos e limites à ferramenta em cada sistema, evitando inconsistências na modelagem relacional.

## 1.2 JUSTIFICATIVA

O Banco de Dados relacional não constitui o meio nativo da orientação a objeto, com relação à persistência do estado dos objetos e existem soluções diferentes para se obter a flexibilidade no desenvolvimento de *software* com o desacoplamento das camadas da aplicação da Gratex. Essa diversidade aparece em soluções como:

a) Mapeamento manual entre os modelos de objetos para relacional, realizado direto no código-fonte e na modelagem relacional;
b) Aplicação do *framework* de persistência para realizar o mapeamento de forma automática entre os modelos de objetos para relacional, ou seja, gerando vínculos e criando tabelas e campos necessários para a persistência do estado dos objetos;
c) Híbrido, mescla das duas soluções acima, a partir da implementação de um *framework* de persistência em sistemas complementares com mapeamento do modelo de objetos para relacional manual e Banco de Dados relacional compartilhado.

Ao atender os projetos de curto prazo com a ausência de *framework* de persistência, a empresa Gratex estabelece o acoplamento entre as suas aplicações e o sistema gerenciador de Banco de Dados, dificultando a manutenção dos sistemas e a migração do SGBD.

Em resumo, a problemática reflete: "*forte acoplamento entre o sistema gerenciador de Banco de Dados e os sistemas legados de uma empresa de manufatura*".

O problema possui características técnicas que podem dificultar e impedir a adoção de estratégias da empresa Gratex como, por exemplo, melhorar o desempenho do desenvolvimento interno de *software*, acelerando a produção de código-fonte e automatização de processos, ou tornar possível a migração do SGBD, por exemplo, sem ou com um mínimo de intervenções dos programadores na aplicação.

## 1.3 OBJETIVO

O objetivo dessa monografia consiste em:

a) Analisar os diagramas de classes e o modelo relacional dos sistemas legados, propondo e efetuando alterações, utilizando como base os padrões de projeto identificados, soluções de problemas recorrentes no desenvolvimento de sistemas orientados a objetos, para facilitar a introdução do *framework* de persistência em sistemas complementares e com Banco de Dados Relacional compartilhado;
b) Aplicar o *Entity Framework* no Estudo de Caso que corresponde a uma parte da aplicação da empresa Gratex e avaliar o Esquema de Banco de Dados gerado. Esta avaliação será sob o ponto de vista do modelo relacional, com base no conceito de normalização, que implica também na análise de dependências funcionais dos campos das tabelas e padrões de projeto;
c) Descrever o caminho para utilização do *framework* de persistência nos sistemas existentes, propondo responsabilidades e limites entre administradores de Banco de Dados, programadores e ferramentas na construção da solução.

Essa pesquisa visa à redução do acoplamento entre as camadas da aplicação da empresa Gratex, manutenção eficiente de código-fonte e a flexibilidade com relação ao SGBD.

## 1.4 CONTRIBUIÇÃO

Visando atender os requisitos de negócio da empresa Gratex, essa pesquisa tem o propósito de identificar as dificuldades e afinidades na utilização de tecnologias de mapeamento de modelos de objetos para o modelo relacional em aplicações existentes que utilizam orientação a objetos e Banco de Dados relacional compartilhado.

Entre as contribuições, destaca-se o trabalho de análise sobre a definição das classes do sistema legado de acordo com padrões de projeto, identificando acertos e falhas, antes da introdução do *framework* de persistência.

Essa pesquisa demonstra fatores relevantes na adoção de um *framework* de persistência em sistemas corporativos existentes com características semelhantes às descritas anteriormente, prestigiando uma solução híbrida onde a ferramenta ora define o mapeamento automaticamente, ora o programador e DBA define o mapeamento manualmente para manter a compatibilidade entre sistemas distintos.

## 1.5 METODOLOGIA

A metodologia empregada tem as seguintes etapas:

a) Estudo do descasamento de impedância entre a tecnologia de orientação a objetos e bancos de dados relacionais;
b) Obtenção e prévia análise dos diagramas de classe e modelo relacional da empresa Gratex;

c) Levantamento bibliográfico sobre modelos de classe UML, modelo relacional, normalização, padrões de projeto e *framework* de persistência;
d) Descrição das características do negócio e sistemas da Gratex;
e) Alteração dos diagramas de classe para o Estudo de Caso;
f) Estudo de Caso - Utilização do *framework* de persistência nos diagramas de classes alterados;
g) Avaliação do Esquema de Banco de Dados gerado pela ferramenta de persistência;
h) Exploração do *framework* de persistência considerando o ambiente proposto de Banco de Dados compartilhado e a sua inserção nos sistemas existentes;
i) Revisão final do texto da monografia;
j) Apresentação da monografia.

O raciocínio adotado nessa metodologia prevalece o entendimento da situação problema e a realização de experimentos para exemplificar a construção da solução que promova a redução do acoplamento entre os sistemas e o SGBD da Gratex.

## 1.6 ESTRUTURA

A organização dessa monografia consiste, no capítulo 2, em uma apresentação do estudo bibliográfico sobre mapeamento objeto-relacional, modelo de classes UML, modelo relacional, normalização, padrões de projeto e *framework* de persistência; em seguida, no capítulo 3, são descritas as características relevantes do sistema legado de uma empresa do segmento têxtil, com o objetivo de proporcionar o entendimento da situação problema; após, no capítulo 4, aplica-se um *framework* de persistência no estudo de caso com a elaboração de um guia para implantação de MOR em sistemas legados e por último, capítulo 5, realiza-se a conclusão e comentários finais sobre esse estudo.

# 2 REGRAS DE MAPEAMENTO DO MODELO OBJETO PARA O MODELO RELACIONAL

No desenvolvimento de programas com orientação a objetos, um dos problemas encontrados, é como realizar a persistência dos objetos da aplicação após a sua criação ou processamento em um meio de armazenamento duradouro.

Dentre as possibilidades, existe a persistência dos objetos através do armazenamento em tabelas do Banco de Dados Relacional que é uma tecnologia amadurecida, amplamente utilizada, suporte disponível no mercado e com diversos fornecedores. Por exemplo, os atributos Nome e Endereço de um objeto Cliente podem ser guardados em campos diferentes de uma tabela e ao instanciar o objeto, os dados são lidos e o preenchimento dos atributos é realizado. No entanto, como essa alternativa não é o meio nativo da orientação a objeto no que se refere à persistência de objetos, a solução demanda um esforço adicional do desenvolvedor em mapear os objetos para um Banco de Dados Relacional. Esse contexto é chamado de mapeamento entre o modelo de objeto para relacional.

Com o objetivo de minimizar os impactos da utilização de Banco de Dados Relacional para a persistência de objetos, nos últimos anos, foram desenvolvidas ferramentas para a automatização do procedimento de mapeamento e abstração do acesso aos dados. Essas ferramentas criam uma interface de programação simples e retiram do desenvolvedor a necessidade de conhecimento aprofundado de SGBD e da linguagem de consulta e manipulação de dados para tais ambientes como a SQL (*Structured Query Language*) (NASCIMENTO, 2009).

O mapeamento entre os modelos de objetos e relacional é a forma de representar objetos da memória em objetos mantidos por um Sistema de Gerência de Banco de Dados (SGBD) e realizar o caminho inverso. Para realizar esse procedimento existem duas maneiras, a primeira, manualmente no código fonte, ou a segunda, através de ferramentas (ROMAN, et al., 2002).

Independente da maneira escolhida, uma questão relevante é o modo diferente como objetos e relações lidam com as associações, originando dois problemas.

Primeiro, há uma diferença na representação, pois objetos lidam com associações armazenando referências (mantidas em tempo de execução por endereços de memória) enquanto que em Banco de Dados Relacionais cria-se uma chave para outra tabela. Em outras palavras, significa que um ou mais campos são incorporados na outra ou na mesma tabela e os dados (valores e não referências) são inseridos, repetido, nesta.

Segundo, os objetos trabalham com coleções para lidar com as referências múltiplas a partir de um único campo, enquanto a normalização (Primeira Forma Normal - 1FN) de Bancos de Dados Relacionais exige que as relações tenham apenas atributos monovalorados, ou seja, para cada tupla da relação, cada atributo apresenta um único valor. Por exemplo, no modelo de objetos, um objeto Departamento evidentemente tem uma coleção de objetos Funcionários que não precisa de qualquer referência de volta ao Departamento. Entretanto, a estrutura no modelo relacional é diferente, uma relação Funcionário inclui um atributo, sob o qual deve ser definida uma restrição de chave estrangeira, para registrar o valor do Departamento associado, já que a relação Departamento não pode ter um atributo multivalorado (DATE, 2005) (SILBERCHATZ, et al., 2006).

Todas as regras de mapeamento entre os modelos considerados são bem definidas e difundidas. Assim, não são apresentadas aqui, mas podem ser facilmente encontradas em (MULLER 2002) e (ELMASRI, 2010).

## 2.1 MODELO DE CLASSES UML

Com a classificação das características e operações similares de objetos diferentes, se torna possível a identificação das classes do sistema, ou seja, a classe captura a estrutura de atributos, operações e relacionamentos comuns de um conjunto de objetos, abstraindo elementos do mundo real. Na UML uma

classe é representada graficamente por um retângulo, visualizado na Figura 1, com três compartimentos para declaração de propriedades da classe. Em um dos compartimentos tem-se o nome da classe; no outro, os atributos e por fim, os métodos. Os sistemas trabalham com instâncias de classes carregadas de dados, as quais originam um objeto (MELO, 2004).

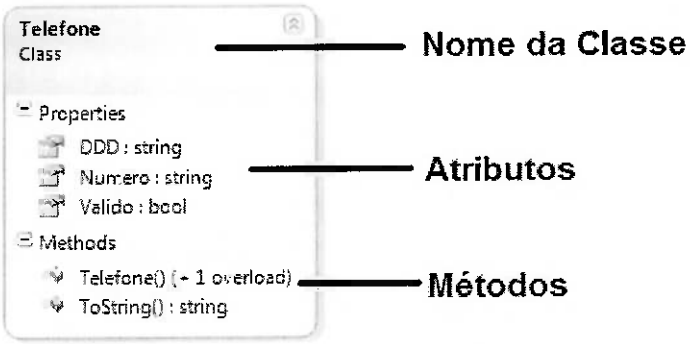


**Figura 1 - Representação de Classe.**

Na definição de atributos e operações de classes pode-se especificar a visibilidade (*public*, *private*, *protected*, *package*) de atributos e operações, tipos de dados (domínio) de cada atributo, escopo (se de classe ou de instância) de atributos e operações, argumentos de operações, tipo de retorno (se existir) para cada operação. O escopo remete a definição, através de um sublinhado no elemento, que os atributos ou métodos são de classe e podem ser obtidos sem instanciar um objeto. Quanto aos estereótipos, notas e restrições não são aqui detalhados, mas informações podem ser facilmente encontradas em (MELO, 2004), além disso, essas características não são contempladas na análise dessa pesquisa.

Para entender o modelo de classes UML e analisar os diagramas de classe da empresa Gratex, é relevante detalhar e definir mais alguns conceitos do paradigma de objetos que podem ser representados num diagrama de classes. Assim, a seguir comenta-se sobre OID (*Object Identifier* - Identificador de objeto) e os termos encapsulamento, interface, herança, polimorfismo e relacionamentos entre classes.

OID remete ao conceito de uma identidade única para todo e qualquer objeto, também denominada referência, definindo a diferença entre objetos

através da sua existência e não pelos valores dos seus atributos (MELO, 2004).

O encapsulamento corresponde à idéia de que os atributos de uma classe só podem ser acessados e atualizados por operações definidas na própria classe, ou seja, uma classe é uma caixa preta, sua estrutura interna (atributos e o código das operações) é encapsulada na própria classe. O objetivo é evitar a dependência da sua implementação interna e proteger o usuário da classe contra mudanças. Para que as alterações futuras ocorram de modo transparente, são criadas interfaces para a classe. Os serviços (ou operações) que uma classe oferece são declarados na sua interface. Outras classes poderão solicitar um serviço enviando uma mensagem (solicitação de execução da operação desejada) para a classe (MELO, 2004).

Com o conceito de herança, também conhecido como generalização, é possível definir um relacionamento especial entre classes que possuem atributos e operações comuns. Nessa estrutura, cria-se uma classe, chamada de superclasse (ou classe genérica) que reúne as propriedades (atributos, operações e até relacionamentos com outras classes) comuns das demais classes. Estas últimas, definidas como subclasses (ou classes-filha ou especializações ou extensões), herdam as propriedades definidas na superclasse e podem estendê-la, ou seja, podem-se declarar outros atributos e operações específicas a subclasse, como também alterar a implementação de uma operação herdada. Em algumas situações, a classe-filha possui mais de uma superclasse e herda os atributos de todos os seus ancestrais, caracterizando a herança múltipla (MELO, 2004). Por exemplo, uma superclasse Veículo pode ter duas subclasses, Carro e Barco, que após uma nova especialização origina uma subclasse chamada Anfíbio, a qual herda características de Veículo, Carro e Barco.

Na Figura 2, a superclasse Veiculo contém os atributos que são herdados pelas classes-filhas Carro e Barco, demonstrando uma herança simples.

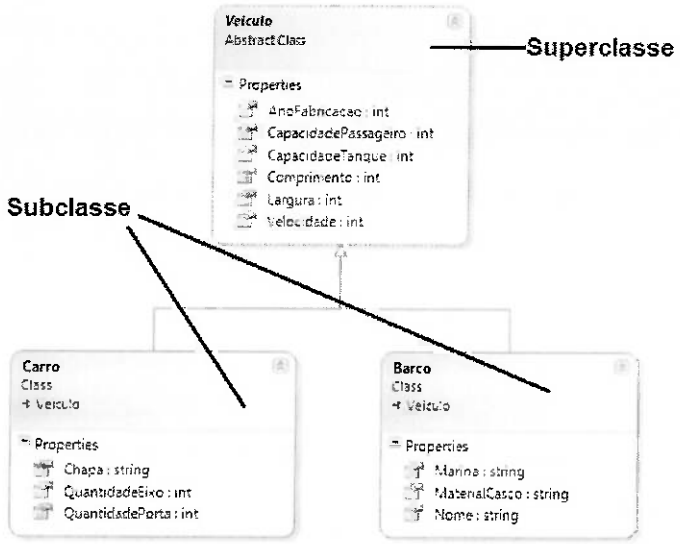


**Figura 2 - Representação de Herança simples.**

Outra característica presente nos modelos de classes UML e associada ao conceito de herança é o polimorfismo. Através desta capacidade se permite definir nas subclasses operações com implementações (métodos) diferentes da superclasse, prevalecendo os métodos descritos nas subclasses. A operação polimórfica é declarada novamente na subclasse, mas com a mesma assinatura, ou seja, com a mesma quantidade e tipo de argumentos, e tipo do retorno que o declarado na superclasse (MELO, 2004).

Na Figura 3 as subclasses Mensalista, Autonomo e Comissionado reescrevem o método CalcularSalario para realizar, cada uma, uma implementação diferente da expressa na superclasse Funcionario que, por sua vez, é uma especialização da superclasse Pessoa. Para uma determinada especialização o salário é fixo (Mensalista), em outra depende da quantidade de horas trabalhadas (Autonomo) e por último o salário é composto de salário base e comissão do mês (Comissionado). Observe que os métodos possuem os mesmos argumentos na assinatura (ano e mês) e possuem o mesmo tipo de retorno, Double, caracterizando um exemplo de polimorfismo.

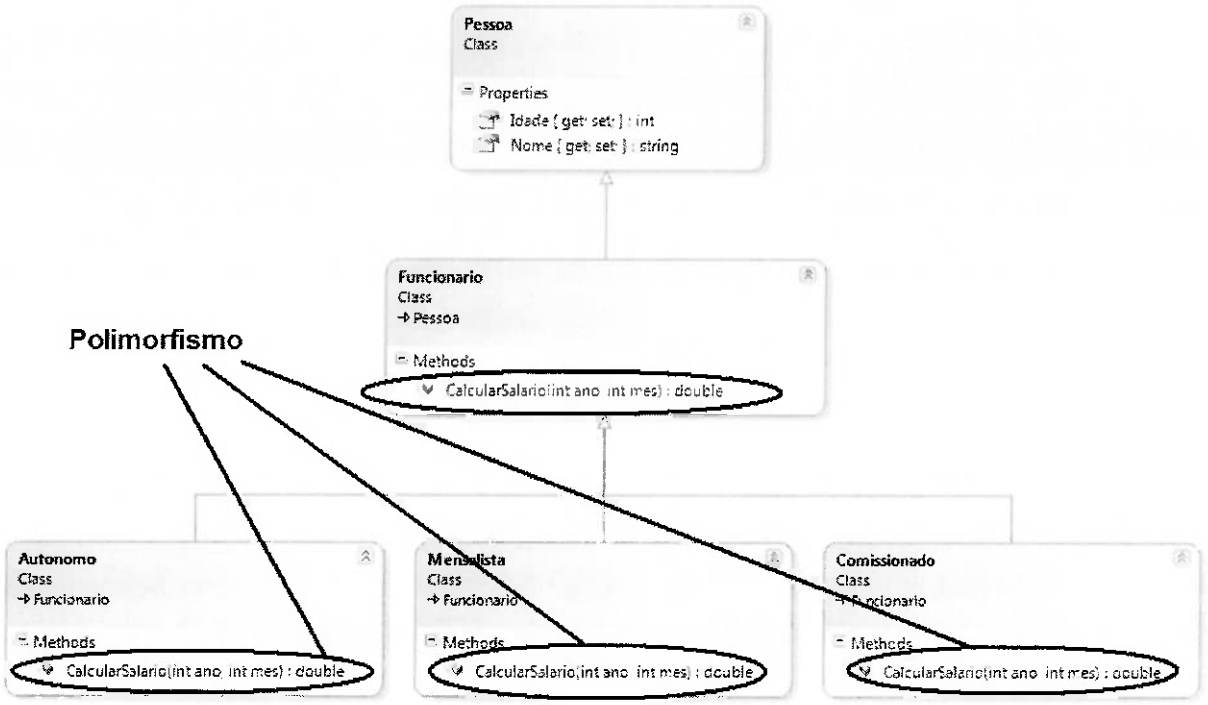


**Figura 3 - Exemplo de Polimorfismo.**

Associações, além da herança que tem características peculiares, podem-se ser utilizadas para representar relacionamentos entre classes. É comum representar a multiplicidade de relacionamentos quando se representa associações. A multiplicidade informa quantas instâncias da(s) classe(s) envolvida(s) podem participar da associação. Tipos especiais de associações são a agregação e a composição. É possível também especificar associações de dependência entre classes, isso significa que a classe dependente é afetada por mudanças da outra classe. Agregação e composição são tipos de associações que definem relacionamentos "todo-parte" (permite a modelagem de objetos complexos), onde a agregação representa um acoplamento fraco, permitindo a classe "parte" participar de outra agregação. Além disso, a remoção de instâncias da classe "todo" não implica na remoção das instâncias da classe "parte" associada. A composição representa um acoplamento forte, uma classe parte só participa de uma composição (só tem uma única classe "todo") e a remoção da instância da classe "todo" implica na remoção das instâncias das classes "parte" associadas (MELO, 2004).

Para mais informações sobre o modelo de classe utilizado nessa pesquisa, indica-se (JACOBSON, et al., 1999), o qual expande, através da UML, as fronteiras da engenharia de software moderna.

## 2.2 MODELO RELACIONAL

O modelo relacional de dados manipula um conjunto de tabelas chamadas de relações, onde os dados são representados por meio de linhas chamadas de tuplas e colunas chamadas de atributos. Cada atributo está associado a um domínio, ou seja, os valores que podem ser registrados em um determinado atributo são definidos pelo tipo de dado atribuído ao atributo, o qual especifica seu domínio. Cada tupla consiste de uma relação entre um conjunto de domínios. Com isso, a relação consiste no subconjunto do produto cartesiano de uma lista de domínios (SILBERCHATZ, et al., 2006).

Em um SGBD Relacional é necessário que existam informações sobre os dados armazenados, tais como (DATE, 2005):

a) Nome da relação: um nome próprio e único dentro do Banco de Dados;
b) Nome do atributo: cada atributo deve ter um nome próprio e único dentro da relação;
c) Tipo de dado do atributo: cada atributo deve ser de um tipo de dado específico (caractere, numérico, data, moeda, entre outros);
d) Tamanho do atributo: cada atributo deve ter um tamanho específico.
e) Chave primária: conjunto de atributos concatenados que possui valor único registrado e que é capaz de identificar uma única tupla da relação. O conceito de chave-primária está vinculado à regra de integridade existencial.
f) Chave estrangeira: conjunto de atributos pertencentes a uma relação que corresponde à chave primária da relação associada. A chave estrangeira é utilizada para representar o relacionamento entre relações. O conceito de chave estrangeira está vinculado à regra de integridade referencial.

Na Figura 4 apresenta-se parte de um esquema de banco de dados, onde se destaca a estrutura do modelo relacional. Nesta, as tabelas (relações) tbl_usuario, tbl_relatorio e tbl_usuario_relatorio são apresentadas. Observe que o conjunto id_usuario e id_relatorio na tabela tbl_usuario_relatorio, consiste na chave-primária desta tabela. Ainda, sobre o campo id_usuario tem-se uma restrição de chave estrangeira referenciando a tabela tbl_usuario, e o mesmo sobre o campo id_relatorio, porém referenciando a tabela tbl_relatorio. Os campos, data_criacao, data_atualizacao e data_inativacao representam uma parte da política interna da empresa Gratex onde, respectivamente, referem-se à criação do registro, a última atualização da tupla e ao mecanismo de deleção não física, ou seja, a aplicação desconsidera registros que tenham o campo diferente do valor nulo. Essa política é utilizada pela empresa para a realização de auditoria interna dos processos.

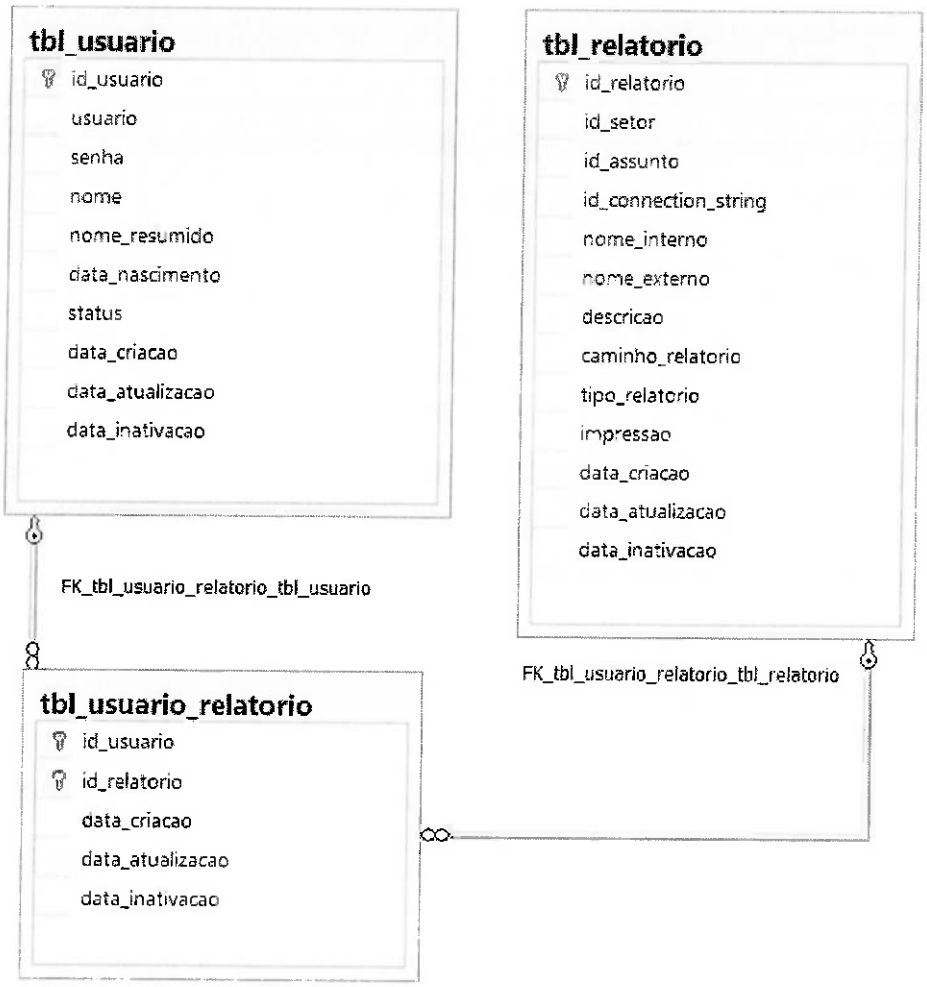


**Figura 4 - Exemplo do Modelo Relacional.**

No que se refere ao mapeamento entre o modelo Entidade-Relacionamento (MER) e o modelo Relacional, existem regras bem difundidas que não serão aqui representadas, mas podem ser facilmente encontradas nas referências (SILBERCHATZ, et al., 2006) (DATE, 2005). No exemplo da figura 4, por exemplo, a tabela tbl_usuario_relatorio é fruto da regra de mapeamento de relacionamentos n:m entre as entidades usuário e relatório de um modelo Entidade-Relacionamento.

### 2.2.1 Normalização

A normalização é uma técnica com o objetivo de eliminar a redundância de dados no Banco de Dados, permitindo a construção de relações que não apresentem anomalias quando as operações de manipulação (inserção, remoção e atualização) dos dados são submetidas ao SGBD.

O conceito de normalização consiste de regras para originar relações bem projetadas, ou seja, com a redução de problemas de manutenção, custo de espaço de armazenamento e problemas de desempenho. A normalização é definida sob o conceito de dependência funcional e Formas Normais cumulativas: 1FN, 2FN, 3FN, BCFN, 4FN e 5FN (DATE, 2005).

Para uma relação estar na primeira forma normal, 1FN, cada tupla deve conter apenas atributos monovalorados (DATE, 2005), por exemplo, na Figura 4 da seção 2.2 as relações contêm apenas um valor para cada atributo.

A dependência funcional consiste no relacionamento entre um conjunto de atributos #X e outro #Y, o qual se refere a uma determinada variável de relação, isto é, se para cada valor de #X existe apenas um #Y, ocorre uma dependência funcional representada por #X → #Y e lida da seguinte forma: #Y depende funcionalmente de #X (DATE, 2005). Por exemplo, considerando o modelo relacional apresentado na Figura 4 da seção 2.2, nota-se uma dependência funcional na relação tbl_usuario, onde o conjunto de atributos {usuario, senha, nome, nome_resumido, data_nascimento, status, data_criacao, data_atualizacao, data_inativacao} depende funcionalmente do atributo {id_usuario}.

Por definição, uma relação está na segunda forma normal se todo atributo que não faça parte da chave é irredutivelmente dependente da chave primária e como as Formas Normais são cumulativas, a relação precisa contemplar a 1FN (DATE, 2005). A relação tbl_relatorio (Figura 4) está na 2FN, pois todos os atributos não chave dependem exclusivamente da chave primária id_relatorio.

Além da necessidade de estar na 2FN, uma relação considera-se na 3FN (terceira forma normal) se todo atributo não pertencente à chave é dependente de forma não transitiva da chave primária, isto é, mutuamente

independentes entre si (DATE, 2005). Na Figura 5, o atributo descricao_localizacao é dependente do atributo localizacao e no caso de uma atualização da descrição de uma determinada localização, será necessário atualizar todos os registros da tbl_pedido, onde a respectiva descrição aparece. Portanto, define-se que a relação tbl_pedido não está na 3FN. Assim, para levar esta relação para a 3FN, será necessário dividi-la em duas relações: tbl_pedido, que não apresentará mais o atributo descricao_localizacao, e tbl_localizacao, com os atributos localizacao (determinante) e descricao_localizacao.

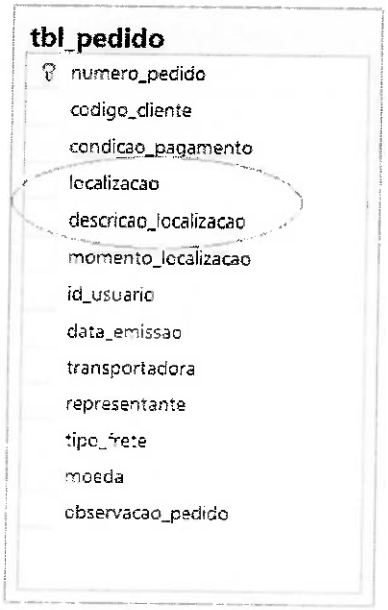


**Figura 5 - Terceira Forma Normal não contemplada.**

No contexto deste trabalho somente serão empregadas a 1FN, 2FN e 3FN, pois na prática, são as que são empregadas. As demais Formais Normais são detalhadas nas referências (DATE, 2005) (SILBERCHATZ, et al., 2006) e apesar de gerarem relações bem projetadas quanto às operações de inclusão, remoção e atualização, também aumentam a probabilidade de junções no acesso aos dados, o que influencia diretamente o tempo de resposta para uma consulta. Junções são operações caras, do ponto de vista de uso de memória e número de acessos a disco, portanto podem afetar diretamente o desempenho do SGBD.

Os bons projetos de um Banco de Dados devem balancear as relações mantendo um bom nível de normalização (pelo menos até a 2FN), de forma a evitar anomalias nas operações de inclusão, remoção e atualização, mas também evitar maior número de junções para a maioria do acesso aos dados.

Outro ponto importante, para verificar a adequação do esquema da base de dados gerado, são as informações sobre a empresa cujo Banco de Dados está sendo modelado, que definem o tipo e freqüência de consultas ao banco de dados. Essas informações ajudarão a definir o nível de normalização adequado a aplicação (SILBERCHATZ, et al., 2006).

## 2.3 PADRÕES DE PROJETO

A partir da experiência adquirida ao longo dos anos, os desenvolvedores de *software* identificaram e documentaram problemas recorrentes no desenvolvimento de sistemas orientados a objetos e passaram a propor soluções que seguem uma mesma linha de raciocínio, ou seja, padrões de projeto. Uma das referências mais conhecidas sobre esse assunto é o livro escrito pelo GOF (*Gang of Four* - Gangue dos Quatro), contendo 23 padrões de projetos (GAMMA, et al., 1994).

Atualmente, existem outros padrões de projetos que facilitam e viabilizam o mapeamento entre o modelo de objeto e relacional através do *framework* de persistência. Na verdade, estes padrões seguem de alguma forma as regras de mapeamento discutidas na seção 2 e não serão aqui detalhados, mas para facilitar a consulta, diversos padrões de projeto foram descritos na seção A.1 do apêndice A.

## 2.4 *FRAMEWORK* DE PERSISTÊNCIA

A camada intermediária de mapeamento do modelo de objeto para o modelo relacional, contendo as regras de persistência, fica situada entre o

modelo de domínio (regras de negócio) e o SGBD. Para esta camada ser considerada um *framework* de persistência, é necessário fornecer recursos com as seguintes características (JOHNSON, 2002):

a) Assegurar integridade dos dados;
b) Geração de *scripts* eficientes;
c) Garantir comportamento correto para acesso por múltiplos usuários;
d) Implementação das regras de negócio independente da estratégia de uso do *framework*, com a possibilidade de trocar a estratégia de persistência sem modificar novamente as regras de negócio;
e) Código fonte de utilização do *framework* necessita ser claro e de fácil manutenção, para concentrar os esforços do programador na definição das regras de negócio.

O desenvolvimento de uma camada com as características descritas, em termos de custo do recurso humano capacitado e de tempo, é um trabalho exaustivo. Exige conhecimento especializado da estrutura e da linguagem de acesso, além de particularidades do SGBD escolhido. A opção pelo uso de um *framework* de persistência de objetos, comercial ou de código livre, que forneça diversos serviços de mapeamento do modelo de objeto para o modelo relacional (LARMAN, 2004), libera o desenvolvedor da necessidade de reter conhecimentos especializados de banco de dados.

Segundo PINHEIRO (2005) a aplicação de um *framework* de persistência em sistemas de informação proporciona benefícios como:

a) Redução do acoplamento entre a camada de aplicação (regras de negócio) e de dados, objeto de estudo dessa pesquisa;
b) Facilidade para uma eventual troca do SGBD;
c) Coleta automatizada de informações das estruturas de classes e objetos do Banco de Dados através da consulta a metadados;
d) Incentivo ao reuso por organizar e agrupar o conhecimento de desenvolvedores em determinado domínio ou aspecto de infra-estrutura, aumentando a produtividade;
e) Simplicidade para manutenção do código-fonte do programa.

O mapeamento e a conversão automatizados entre o modelo objeto e o modelo relacional realizado por um *framework* de persistência, simplesmente correspondem a ter a mesma funcionalidade com menos código-fonte, do que se comparado a comandos SQL embutidos no código-fonte, elaboração de procedimentos armazenados no SGBD com comandos SQL (*stored procedures*) ou outra forma de acesso e manipulação dos dados; e a evitarem acesso à camada de armazenamento duradoura através de um dispositivo de memória temporária (*cache*) transparente dos objetos, melhorando o desempenho do sistema. No entanto, as ferramentas de mapeamento entre os modelos de objeto e o relacional, por realizarem constantemente a consulta de metadados, também podem onerar o desempenho da aplicação se não utilizadas considerando essa importante característica (MEHTA, 2008).

### 2.4.1 *Entity Framework*

Durante anos o mapeamento do modelo de objeto para o modelo relacional tem sido pouco explorado pelas áreas de desenvolvimento que utilizam tecnologia da indústria de *software* Microsoft. No entanto, com o passar do tempo e em função da necessidade de uma camada de persistência dos dados reutilizável e de fácil manutenção, diversas ferramentas para realizar esse tipo de mapeamento apareceram no mercado, como por exemplo, o *NHibernate*, cujo esquema pode ser visualizado na Figura 6.

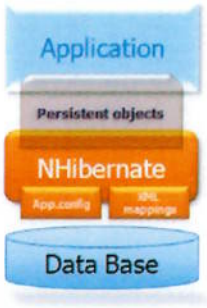


**Figura 6 - *NHibernate*, camada entre Aplicação e Banco de Dados.**
**Fonte: (NHIBERNATE, 2010).**

A Microsoft, após superar o seu primeiro obstáculo no que se refere ao desenvolvimento de *software* com o uso do *framework* .NET e um ambiente maduro de programação orientado a objetos, abriu espaço para prover uma camada de persistência dos dados, *Entity Framework*, como recurso nativo no seu ambiente integrado de desenvolvimento (IDE), o *Visual Studio*.

O segundo obstáculo foi superado no *Visual Studio* 2008, com a introdução do *Entity Framework*, finalmente estabelecendo os recursos necessários para construção modular e reutilizável da camada de persistência dos dados (MEHTA, 2008).

A ferramenta de mapeamento do modelo de objeto para o modelo relacional a ser considerada nessa pesquisa é o *Entity Framework* do .NET *framework* 4.0 e nativo no IDE do *Visual Studio* 2010 da Microsoft.

Uma das características do *Entity Framework* é o suporte a classes POCO, as quais definem seus atributos sem herdar ou conter informações específicas sobre uma tecnologia ou um *framework* de persistência.

Na primeira versão do *Entity Framework*, isso não era suportado, mas na versão atual, existem classes de domínio simples e que podem ser utilizadas com outras tecnologias. Essas classes são equivalentes as classes POJO do *Java* (PAULI, 2010).

Um dos fatores relevantes para escolha do *Entity Framework* como ferramenta, refere-se à recente aquisição do ambiente de desenvolvimento integrado *Visual Studio* 2010 pela Gratex, o qual será utilizado na criação e manutenção dos *softwares* da empresa. Como a principal contribuição da presente pesquisa está na avaliação e preparação do ambiente ao longo da introdução de um *framework* de persistência para uso pela referida empresa, entende-se que é a escolha mais apropriada para este trabalho, justificando-se assim, a ausência de comparações com as outras opções disponíveis no mercado.

# 3 A EMPRESA GRATEX

A Gratex é uma empresa de manufatura nacional do segmento têxtil que oferece ao mercado um mix de produtos, nacional ou importado, do básico ao inovador, focando nos segmentos feminino, masculino e infantil. O produto da Gratex divide-se em tecido plano ou malharia, e sua coleção é alinhada com o que acontece de mais atual na indústria da moda.

Com uma equipe de desenvolvedores de *software*, composta por Analista de Sistemas de Internet, Administrador de Banco de Dados, Analista de Projeto e Sistemas e Analistas Programadores (.NET e ADVPL[3]), a empresa trabalha na manutenção e evolução dos sistemas internos como *e-commerce*, *BackOffice* e na customização do ERP Microsiga, sistema integrado de gestão empresarial do grupo TOTVS.

Situada entre as grandes empresas (MORAES, 2010) do seu ramo de atuação, considerando faturamento bruto, os projetos de desenvolvimento de *software* da Gratex buscam contemplar requisitos não-funcionais como usabilidade e desempenho nas aplicações e atender com agilidade a mudança constante nos requisitos das áreas de negócio.

Em uma visão macro, comparando a demanda de projetos e as manutenções solicitadas, destaca-se a área comercial da empresa Gratex que utiliza como ferramenta de trabalho o sistema *e-commerce*, GratexNet, e o sistema *BackOffice*, GratexOffice.

## 3.1 ESPECIFICAÇÃO DA APLICAÇÃO ESCOLHIDA - ESTUDO DE CASO

O Estudo de Caso desta pesquisa é composto por partes de sistemas distintos. Uma das partes corresponde à emissão de pedido de venda pelo sistema GratexNet e a outra por uma aprovação deste pedido no *BackOffice.*

[3] ADVPL, sigla de *Advanced Protheus Language*, é a linguagem de programação nativa do sistema ERP Microsiga do grupo TOTVS.

Dentre estes sistemas, o mais antigo (aproximadamente seis anos desde a sua concepção inicial), *BackOffice*, contém diversas classes que não seguem padrões de projeto e relações que não seguem o mínimo de normalização praticada no desenvolvimento de *software*, somente a 1FN. Isto reflete em classes como, por exemplo, a classe CPedido contendo mais de 80 atributos, onde após uma análise prévia e grosseira, a quantidade necessária real poderia ser pelo menos metade destes atributos. Uma análise mais detalhada levará com certeza a classes mais bem projetadas, com número reduzido e necessário de atributos.

A evidência de não seguir as melhores práticas referentes aos padrões de projeto do modelo de objetos e a normalização do modelo relacional, não necessariamente indicaram imediatamente a existência de problemas na aplicação *BackOffice*. No entanto, ao longo das manutenções e melhorias deste sistema ficaram nítidas as dificuldades criadas pela ausência de boas práticas, por exemplo, devido ao forte acoplamento observou-se acréscimo no tempo de realização de uma demanda de TI que necessitava alterar a estrutura de classes e/ou relações.

Com este ambiente sem uma preparação adequada, o uso de um *framework* de persistência poderá onerar o desempenho da aplicação, ao invés de colaborar com a redução do acoplamento entre o SGBD e os sistemas existentes na empresa. Aliás, cabe ressaltar que mesmo sem o uso de um *framework*, o trabalho dos desenvolvedores também seria oneroso em termos de tempo para implementação da base de dados e principalmente na fase de manutenção. Isto sem considerar o desempenho do SGBD quando consultas que modifiquem a base de dados forem submetidas, dado o tamanho esperado das tabelas e redundância excessiva de dados.

Para propor uma solução a essa problemática vivenciada na Gratex, a aplicação com a situação mais crítica, *BackOffice* foi elencada para uma revisão das classes e geração do Banco de Dados Relacional, tendo como objetivo principal a avaliação e preparação do ambiente para introdução de um *framework* de persistência.

Os ganhos obtidos com a adoção do *framework* de persistência foram abordados anteriormente, item 2.4, e não serão aqui descritos novamente.

## 3.2 DIAGRAMAS DE CLASSES - PANORAMA ATUAL

O fato do *Visual Studio* 2010 conter semelhanças visuais entre os modelos gerados, *ClassDiagram* e *Entity Data Model*, este último relacionado ao uso do *Entity Framework*, torna natural a sua escolha como ferramenta para realizar o levantamento do panorama atual de diagramas de classes.

Para entendimento do modelo de classes UML a seguir, define-se que as classes e tipos de atributos padrões da linguagem de programação *Visual Basic* .NET como *DataTable*, *Boolean*, *String*, *Char*, *Date*, *Integer* e *Double*, utilizados no desenvolvimento da aplicação *BackOffice*, não são foco desta pesquisa e devem ser suprimidos por exemplo, da análise de dependência entre as classes.

No sistema *BackOffice* da Gratex existem aproximadamente 80 classes, porém a escolhida para análise no diagrama de classes será a de nome CPedido. Como esta classe é parte integrante da tela de aprovação do pedido de venda do GratexNet ou Microsiga, processo importante da empresa, e não está normalizada ou seguindo padrões de projeto, utiliza-se a CPedido com o objetivo de proporcionar uma contribuição à empresa, caso sejam identificadas possíveis alterações nesta análise acadêmica. A ausência de normalização ou padrões de projeto, não necessariamente caracteriza um problema, mas pode indicar um caminho a seguir em uma análise sobre o diagrama de classes. De acordo com as regras de negócio do sistema, é importante ressaltar que esta classe tem alguns nomes de atributos com o sufixo "Original", os quais, para efeito de histórico, representam um estado do objeto antes da aprovação ou reprovação do pedido de venda por um coordenador de vendas ou gerente comercial.

Outro ponto importante é o ciclo de desenvolvimento da classe escolhida, a qual não teve um planejamento e tempo de criação adequado. Isso não necessariamente pode justificar o panorama atual a ser apresentado, no entanto pode reconstruir parte do cenário de desenvolvimento, no qual os programadores elaboraram o diagrama de classes.

Como comentado anteriormente (seção 3.1), a classe CPedido possui 85 atributos. Assim, a apresentação desta classe completa numa figura seria

de difícil visualização, além do número excessivo de atributos, existem ainda três relacionamentos de dependência com outras classes. Desta forma, optou-se por uma simplificação que não afetará a análise foco dos objetivos desta monografia. A Figura 7 apresenta parte do diagrama de classes que representa a aplicação *BackOffice*, o diagrama mostra uma simplificação da classe CPedido e suas associações. Conforme visualizado, o diagrama de classes possui duplicidade de enumeração, ausência de classes para o agrupamento de atributos semanticamente pertencentes ao mesmo objeto, e outros problemas a serem abordados na seção 3.2.1.

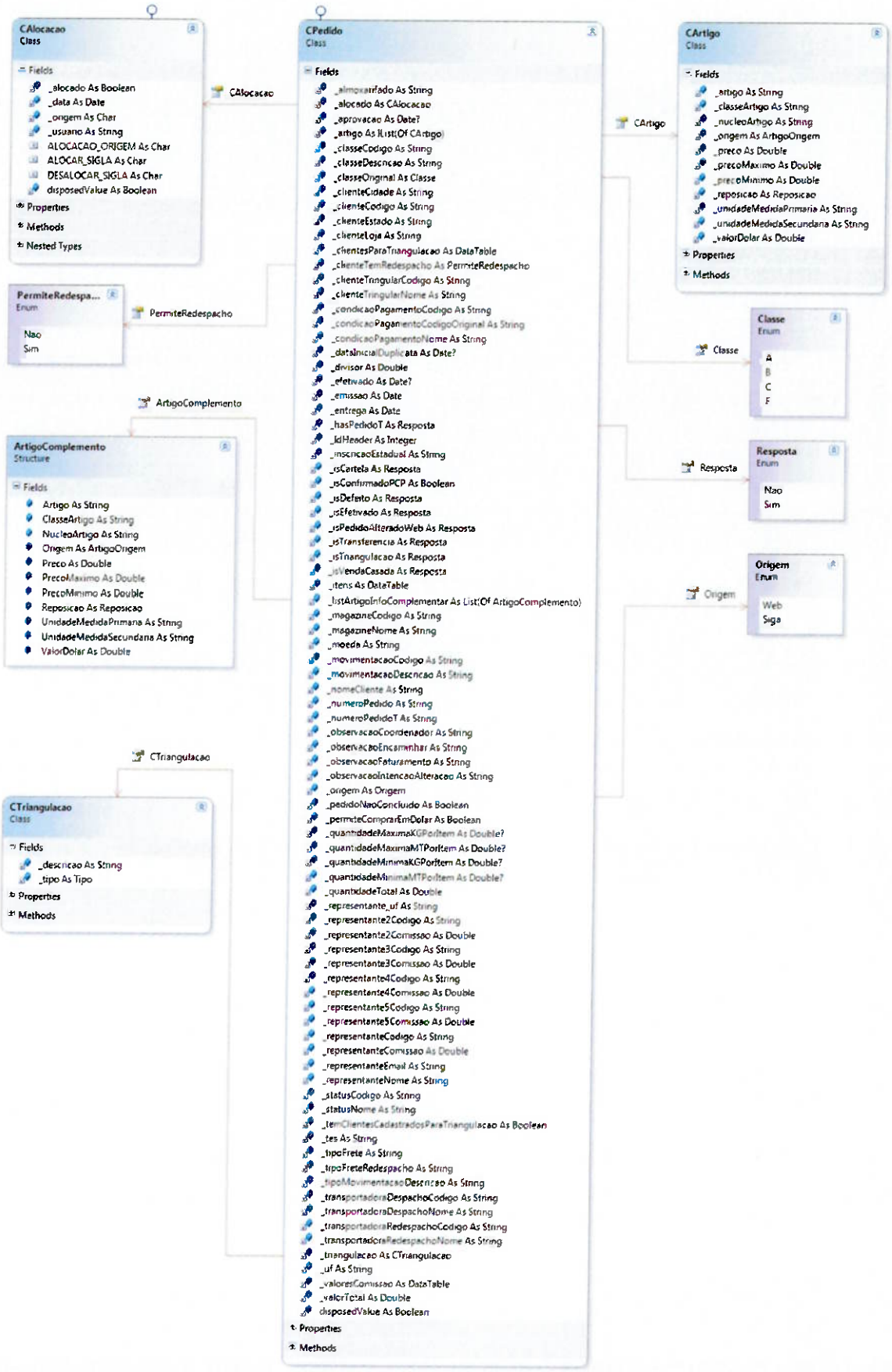


**Figura 7 - Diagrama de Classes da Aprovação do Pedido no *BackOffice*.**

### 3.2.1 Avaliação do Diagrama de Classe

Com base no diagrama de classes apresentado, é possível elencar problemas da classe CPedido e suas respectivas soluções:

a) Duplicidade de enumeração: "Resposta" e "PermiteRedespacho" podem ser representadas apenas por uma única enumeração para facilitar a manutenção futura. Exemplo de alteração: manter apenas "Resposta".

b) Ausência de normalização e regras do modelo de classes UML: Os atributos _transportadoraDespachoCodigo e _transportadoraDespachoNome conforme a definição do modelo de objetos pode originar uma classe Transportadora, a ser utilizada também nos atributos relacionados ao redespacho (_transportadoraRedespachoCodigo e _transportadoraRedespachoNome), e serem trocados por um único atributo de despacho do tipo Transportadora (_transportadoraDespacho). Os atributos _classeCodigo e _classeDescricao sugerem a criação de um único atributo _classe do tipo ClasseProduto, do mesmo modo que o atributo _classeOriginal. Com relação aos atributos _clienteCodigo, _nomeCliente, _clienteTemRedespacho, _clienteLoja, _permiteComprarEmDolar, _clienteEstado, _clienteCidade e _inscricaoEstadual após classificá-los segundo suas características (ver seção 2.1), é possível agrupá-los em uma classe Cliente e colocar apenas um atributo _cliente com este tipo na CPedido, sendo que o mesmo precisa ocorrer com o cliente triangular, ou seja, eliminar os atributos _clienteTriangularCodigo e _clienteTriangularNome, e criar um atributo _clienteTriangular do tipo Cliente. Nesta análise é realizada a exclusão do atributo _uf, o qual correspondia ao mesmo conteúdo do atributo _clienteEstado. Continuando o agrupamento por característica, os atributos _condicaoPagamentoCodigo e _condicaoPagamentoNome formam a classe CondicaoPagamento e geram-se dois novos atributos deste mesmo tipo: _condicaoPagamento e _condicaoPagamentoOriginal (elimina-se _condicaoPagamentoCodigoOriginal). Ainda neste mesmo

raciocínio, substitui-se _magazineCodigo e _magazineNome por um atributo _magazine do tipo Cliente. Com o agrupamento dos atributos _movimentacaoCodigo, _movimentacaoDescricao e _tipoMovimentacaoDescricao constitui-se a classe Movimentacao e coloca-se na CPedido o atributo deste tipo com o nome de _movimentacao. Os atributos _representante_uf, _representanteEmail, _representanteNome, _representanteCodigo, _representanteComissao, _representante2Codigo, _representante2Comissao, _representante3Codigo, _representante3Comissao, _representante4Codigo, _representante4Comissao, _representante5Codigo e _representante5Comissao definem uma nova classe chamada Representante e adicionam-se na classe CPedido, atributos deste tipo com os seguintes nomes: _representante1, _representante2, _representante3, _representante4 e _representante5. Por último, agrupa-se _statusCodigo e _statusNome na classe Status e coloca-se apenas o atributo deste tipo, chamado _status, na CPedido.

c) Listas associadas à classe CPedido sem a definição de uma classe específica: por exemplo, o atributo _itens pode ser do tipo ItemPedido e não um *DataTable*, o qual é uma classe padrão da linguagem de programação. Sem a definição de um tipo específico as propriedades visíveis serão as do *DataTable* e isso dificulta a manipulação e obtenção de dados da lista de itens do pedido associada. Além disso, os atributos _quantidadeTotal e _valorTotal podem ser eliminados da CPedido e tornarem se propriedades públicas, somente leitura, da lista de itens do pedido de venda. Outro caso semelhante é do atributo _clientesParaTriangulacao que pode ser uma lista do tipo Cliente.

d) Atributos não relacionados diretamente ao cabeçalho do pedido de venda: por exemplo, os atributos com a quantidade limite por unidade de medida do item de pedido _quantidadeMaximaMTPorItem, _quantidadeMaximaKGPorItem, _quantidadeMinimaMTPorItem, _quantidadeMinimaKGPorItem podem ser propriedades, somente leitura, como mínimo e máximo permitido do atributo _itens ou uma classe específica de validação a ser consultada na aprovação do pedido.

Avaliando o diagrama de classes, entende-se uma necessidade de alteração nas classes para atender as regras do modelo de objetos (UML) e do modelo relacional, adequando o sistema a introdução de uma ferramenta que realizará a persistência de objetos em SGBD relacional.

## 3.3 MODELO DE CLASSES PROPOSTO

Na elaboração do modelo de classes proposto, através do *Visual Studio* 2010, além das mudanças realizadas através do próprio diagrama (arquivo com extensão "cd"), também foram criadas classes e realizadas alterações diretamente no código fonte da aplicação, as quais eram refletidas automaticamente ao se carregar o modelo de classes, demonstrando uma vantagem de se utilizar um ambiente integrado de desenvolvimento.

Segue na Figura 8, exemplo de classe criada na linguagem de programação *Visual Basic* .NET e, com o objetivo de simplificar o entendimento, neste caso a classe não apresenta o algoritmo que preenche seus atributos privados.

```
Public Class Transportadora

    Private _transportadoraCodigo As String = String.Empty
    Private _transportadoraNome As String = String.Empty

    Private Sub New()
        MyBase.New()
    End Sub

    Public Sub New(ByVal codigo As String)
        Me.New()
    End Sub

    Public ReadOnly Property Codigo() As String
        Get
            Return (Me._transportadoraCodigo)
        End Get
    End Property

    Public ReadOnly Property Nome() As String
        Get
            Return (Me._transportadoraNome)
        End Get
    End Property

End Class
```

**Figura 8 - Código da classe Transportadora criada para o modelo proposto.**

Após identificar diversos pontos a serem melhorados no diagrama de classes, com o objetivo de facilitar a inserção e uso do *framework* de persistência, segue na Figura 9 um modelo proposto colocando em prática a criação de mais sete classes (CondicaoPagamento, Cliente, ClasseProduto, Movimentacao, Representante, Status e Transportadora) e a eliminação da duplicidade de enumeração, ambas descritas na seção 3.2.1. Nesta primeira alteração, destaque para a eliminação de 25 atributos, reduzindo de 85 para 60 atributos, com a possibilidade de redução maior ao aplicar todas as outras soluções.

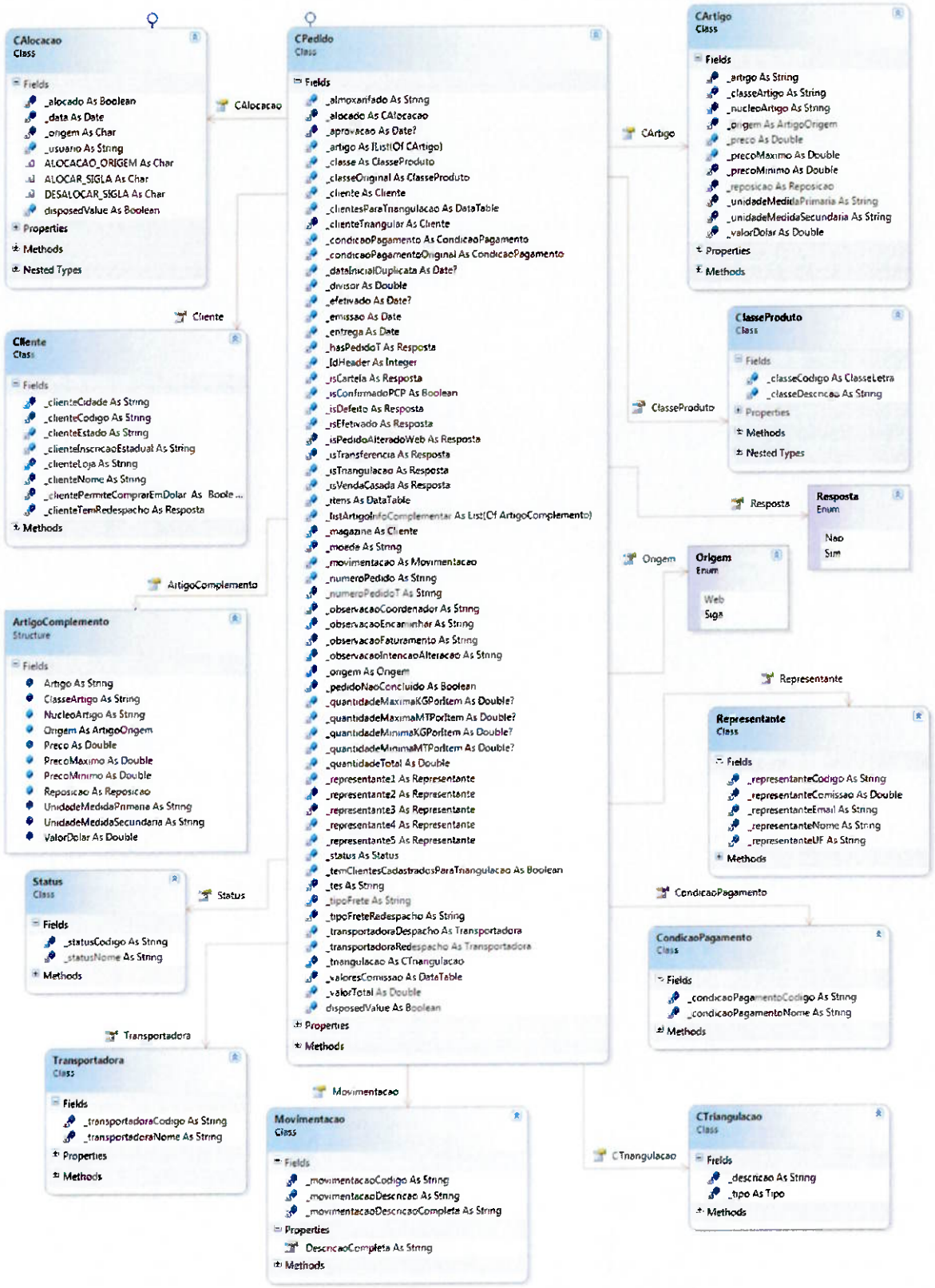


**Figura 9 - Diagrama de Classes Proposto do *BackOffice*.**

Por fim cabe ressaltar que as modificações propostas, produto de uma revisão de parte do sistema existente, não foram exaustivas. Como observado,

houve uma redução em torno de 30% no número de atributos na classe CPedido e, em uma revisão mais detalhada, essa redução seria maior. O objetivo aqui foi mostrar que para o caso em questão, e outros que apresentem problemas similares aos já destacados, a importância da revisão, caso contrário a base de dados pode ficar repleta de redundâncias desnecessárias, sem falar em possíveis inconsistências. Além disso, uma base de dados sem um nível de normalização adequado resultará em baixo desempenho para acessos que modifiquem a base de dados. O quadro agrava-se quando se opta pelo uso de uma ferramenta de persistência, a qual se baseia exclusivamente nas classes existentes para gerar a base de dados. Classes bem definidas conduzem a geração de uma base de dados no mínimo na 2FN, reduzindo a necessidade de grandes modificações na mesma devido a questões de desempenho.

# 4 APLICAÇÃO DO *ENTITY FRAMEWORK* NO CASO ESCOLHIDO

Neste capítulo pretende-se descrever a utilização do *framework* de persistência. Como o caso apresentado na seção 3.2 é, mesmo com as simplificações já realizadas, complexo para o objetivo deste capítulo, optou-se pelo uso de um modelo de classe mais simplificado para facilitar o entendimento da ferramenta e manter o foco no uso dos seus recursos.

Diferente da demonstração realizada até este momento, onde são feitas as modificações no diagrama de classes existente, optou-se por gerar o modelo de entidade de dados, necessário para a utilização do *Entity Framework*, a partir das tabelas existentes no banco de dados. Isto porque, no caso da empresa Gratex, o modelo relacional está estabelecido pelo sistema ERP e o objetivo é introduzir o uso da ferramenta no sistema legado, GratexOffice.

Ao longo deste capítulo sobre a aplicação do *framework* de persistência, realizam-se comparações quando apropriado com o caso escolhido e vínculos com a teoria de padrões de projeto, normalização, modelo relacional ou objeto, expondo contribuições desta pesquisa e benefícios da utilização da ferramenta pela empresa Gratex.

## 4.1 MODELO DE ENTIDADE DE DADOS - *ENTITY DATA MODEL*

Antes de iniciar o uso do *Entity Framework*, gera-se um modelo representativo dos dados, o *Entity Data Model* (EDM), o qual é necessário para manutenção dos relacionamentos e modificações do esquema do banco de dados através da interface gráfica do *Visual Studio* 2010. O EDM ou modelo de entidades de dados pode ser considerado o principal elemento do *Entity Framework*, onde: em tempo de desenvolvimento existe um arquivo de

extensão "edmx" e em tempo de execução existem três arquivos XML[4], CSDL (*Conceptual Schema Definition Language*) - definição do modelo conceitual, MLS (*Mapping Specification Language*) - definição do mapeamento e SSDL (*Store Schema Definition Language*) - responsável pela persistência, conforme pode ser visualizado na Figura 10. Um EDM divide-se em (MACORATTI, 2009):

a) Entidades - instâncias de tipos de entidades como Usuario e Relatorio;
b) Relacionamentos - instâncias de tipos de relacionamentos, os quais são associações entre dois ou mais tipos de entidades.

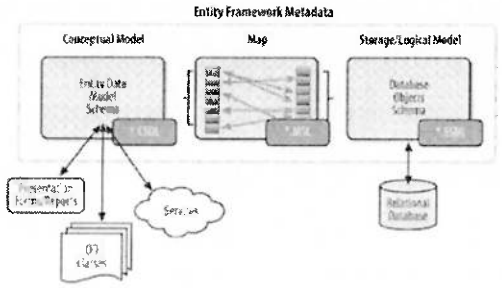


**Figura 10 - *Entity Data Model* em tempo de execução.**
**Fonte: (MACORATTI, 2009).**

Para obter mais informações sobre o passo a passo da geração do arquivo com extensão "edmx" contendo o modelo de entidade de dados, consulte a seção B.1 do apêndice APÊNDICE B - *ENTITY FRAMEWORK*.

Na Figura 11 é possível visualizar o EDM de uma parte do sistema GratexOffice, o qual compreende o cadastro de usuário e relatório, as permissão de usuário x relatório e o cabeçalho resumido do pedido de venda da empresa. Neste caso, o atributo id_usuario da entidade Pedido corresponde ao usuário que aprova o pedido de venda digitado no GratexNet.

[4] XML, sigla de *Extensible Markup Language*, é uma linguagem de marcação para facilitar o compartilhamento de dados através da Internet (W3C, 2008).

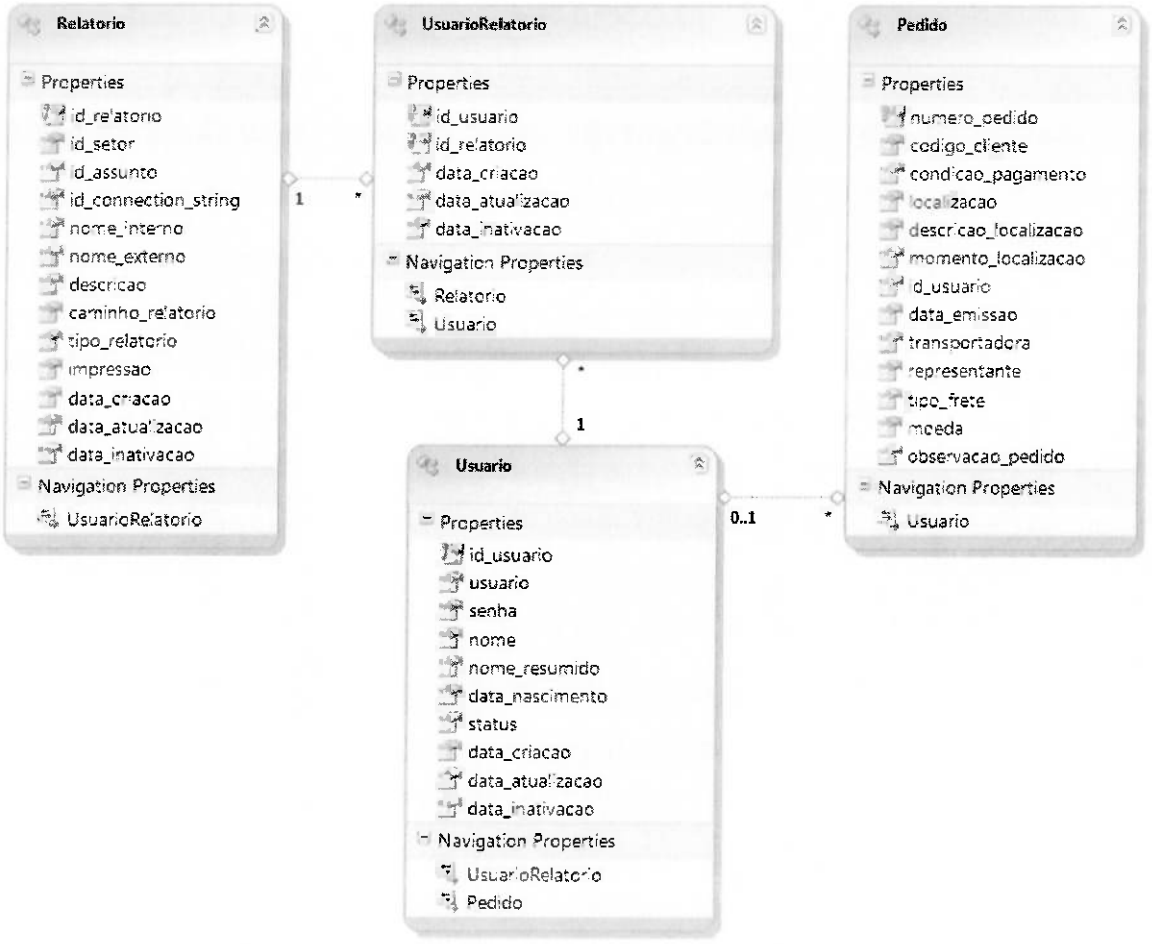


**Figura 11 - Modelo denominado *Entity Data Model*.**

Como o modelo de entidade de dados, EDM, foi gerado a partir de tabelas do modelo relacional, a ausência de normalização, demonstrada na Figura 5 da seção 2.2.1, volta a ser apresentada na entidade Pedido da Figura 11. Com isto, evidencia-se a necessidade de uma revisão do modelo relacional que está sendo vinculado às classes pelo *Entity Framework*, a qual pode utilizar os conceitos de normalização apresentados na seção 2.2.1.

Neste momento é importante destacar a importância do modelo relacional com um bom nível de normalização, ao menos até a 2FN, e relacionamentos fisicamente definidos através de chaves estrangeiras para se obter a geração do modelo representativo que proporcione uma visualização macro adequada das entidades normalizadas e seus fluxos de navegação, *Navigation Properties*, estabelecidos. Outro ponto a ser ressaltado é a presença dos padrões de projeto estruturais como campo identificador (id_relatorio, id_usuario), mapeamento de chave estrangeira que pode ser

observado na criação das propriedades de navegação (*Navigation Properties*) e mapeamento de tabela associativa (entidade UsuarioRelatorio) detalhados na seção A.1 do apêndice A.

## 4.2 MAPEAMENTO DO MODELO RELACIONAL PARA O *ENTITY DATA MODEL* (EDM)

Para a elaboração do mapeamento do modelo relacional para o *Entity Data Model*, seleciona-se a entidade desejada e através de recursos gráficos do *Visual Studio* ("*Mapping Details*") é possível definir de forma amigável qual propriedade da entidade ("*Value / Property*") corresponde a um atributo ("*Column*") da tabela. Note que é possível existir mais de uma relação do modelo relacional por entidade do modelo conceitual e uma coleção de condições por relação utilizada no mapeamento, demonstrando a flexibilidade do *Entity Data Model* na Figura 12.

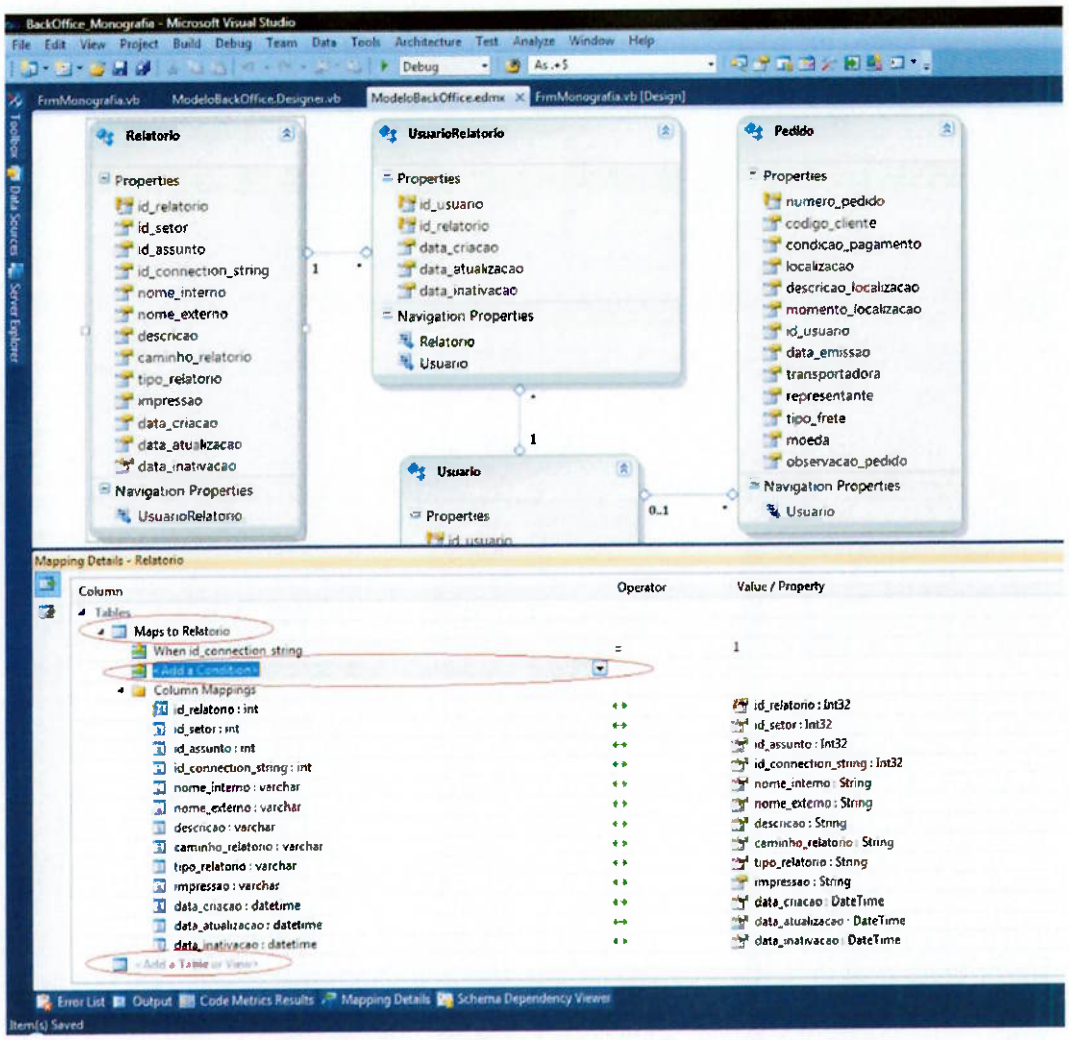


**Figura 12 - Mapeamento relacional x objeto do *Entity Data Model*.**

Após definir o mapeamento, o programador pode utilizar as entidades do modelo representativo no código fonte de suas aplicações através de uma sintaxe unificada de métodos chamada de *Language Integrated Query*, LINQ, funcionalidade composta de expressões *lambda*[5], tipos anônimos e operadores de consulta, por exemplo: *Select*, *Where*, *GroupBy*, *OrderBy*, *Distinct*, entre outros comandos característicos da linguagem de consulta e manipulação de dados SQL.

[5] Expressão *lambda* refere-se a uma função anônima com expressões e instruções que usam o operador *lambda* "=>", lido como "vai para", onde o lado esquerdo do operador *lambda* especifica os parâmetros de entrada (se houver) e o direito contém a expressão ou o bloco de instruções (MSDN, 2011).

## 4.3 UTILIZAÇÃO DO *ENTITY FRAMEWORK*

Um dos recursos disponíveis para utilização no *Entity Framework* é a criação de uma base de dados a partir do EDM, a qual está descrita com mais detalhes na seção B.2 do apêndice B.

Com a criação das relações na base de dados e o uso das entidades do *Entity Framework*, é possível desenvolver uma tela para consulta dos dados, Figura 13, através de uma quantidade reduzida de linhas de código no *Visual Basic* .NET. Se comparado com o desenvolvimento de outras telas do *BackOffice*, este procedimento é mais rápido e de fácil manutenção, pois a cada execução não é necessário criar uma conexão com a base de dados e enviar, por exemplo, um comando SQL ou desenvolver uma *stored procedure* específica. Apenas com recursos do *Entity Framework*, os dados são consultados via LINQ. Para utilizar efetivamente as entidades e obter o benefício de deixar o sistema independente do fabricante de banco de dados ou linguagem de consulta, utiliza-se o LINQ para escrever os comandos enviados à base de dados, conforme visualizado na Figura 14.

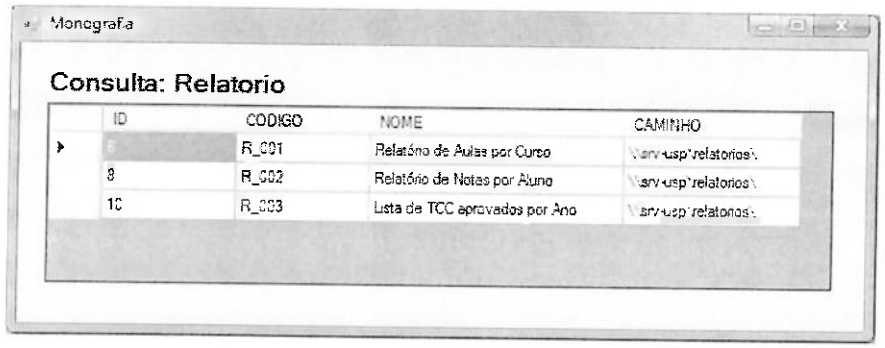


**Figura 13 - Tela de consulta aos dados da entidade Relatorio.**

```
Private Sub CarregarDadosGridView()

    Dim entidade As New DBTCCEntidades()
    Dim resultado As Object

    resultado = From p In entidade.Relatorio
                Select ID = p.id_relatorio,
                    CODIGO = p.nome_interno,
                    NOME = p.nome_externo,
                    CAMINHO = p.caminho_relatorio

    lblNomeConsulta.Text = "Consulta: " + entidade.Relatorio.EntitySet.Name

    dgvDados.DataSource = resultado

End Sub
```

LINQ

**Figura 14 - Código LINQ acessando entidade do EDM.**

Outra característica importante do *Entity Framework* que traz muita agilidade ao desenvolvimento das aplicações é a facilidade de atualização dos dados através da manipulação direta de objetos e o uso de métodos prontos como "*AddObject*", "*DeleteObject*" e o "*SaveChanges*", conforme demonstrado na Figura 15.

```
Private Sub btnAdicionarItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnAdicionarItem.Click

    Dim entidade As New DBTCCEntidades()
    Dim relatorio As New BackOffice_Monografia.Relatorio()

    relatorio.id_relatorio = 1
    relatorio.id_setor = 1
    relatorio.id_assunto = 1
    relatorio.id_connection_string = 1
    relatorio.nome_interno = "R_004"
    relatorio.nome_externo = "Relatório NOVO"
    relatorio.descricao = "Relatório NOVO"
    relatorio.caminho_relatorio = "\\srv-usp\relatorios\"
    relatorio.tipo_relatorio = "2"
    relatorio.data_criacao = Now
    If entidade.Relatorio.Where(Function(p) p.nome_interno = "R_004").Count() = 0 Then

        entidade.Relatorio.AddObject(relatorio)
        entidade.SaveChanges()

    End If
    CarregarDadosGridView()

End Sub

Private Sub btnExcluirItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnExcluirItem.Click

    Dim entidade As New DBTCCEntidades()
    Dim relatorio As New Object

    If entidade.Relatorio.Where(Function(p) p.nome_interno = "R_004").Count() = 1 Then
        relatorio = entidade.Relatorio.First(Function(p) p.nome_interno = "R_004")

        entidade.DeleteObject(relatorio)
        entidade.SaveChanges()

    End If
    CarregarDadosGridView()

End Sub
```

**Figura 15 - Código relacionado à persistência de um objeto.**

Para realizar pesquisas e retornar objetos com o *Entity Framework*, pode-se também utilizar o LINQ, o qual possui a instrução *Where* para filtrar os objetos desejados. Na Figura 16, segue exemplo de uma pesquisa sobre os relatórios que possuem no atributo nome_externo a expressão "TCC" e que o atributo descricao esteja preenchido.

```
Private Sub btnPesquisarTCC_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnPesquisarTCC.Click

    Dim entidade As New DBTCCEntidades()
    Dim resultado As Object

    resultado = From p In entidade.Relatorio
                Where p.descricao <> String.Empty _
                And p.nome_externo.Contains("TCC")
                Select ID = p.id_relatorio,
                    CODIGO = p.nome_interno,
                    NOME = p.nome_externo,
                    CAMINHO = p.caminho_relatorio

    lblNomeConsulta.Text = "Consulta: " + entidade.Relatorio.EntitySet.Name

    dgvDados.DataSource = resultado

End Sub
```

**Figura 16 - Código para realizar pesquisa com o *Entity Framework*.**

Na Figura 17 é possível visualizar um exemplo de *script* na sintaxe SQL, criado em poucos passos através do *Entity Framework*, mais detalhes na seção B.2 do apêndice B, para a construção da base de dados que irá armazenar os objetos mapeados nas entidades de dados do EDM. É importante notar que nesta geração não temos opções sobre o que está sendo gerado então, caso seja necessário realizar alguns ajustes, por exemplo, não considerar alguma entidade, isso terá que ser feito diretamente no arquivo criado.

```
-- ----------------------------------------------------
-- Entity Designer DDL Script for SQL Server 2005, 2008, and Azure
-- ----------------------------------------------------
USE [DBTCC];
GO
IF SCHEMA_ID(N'dbo') IS NULL EXECUTE(N'CREATE SCHEMA [dbo]');
GO
-- Dropping existing FOREIGN KEY constraints
IF OBJECT_ID(N'[dbo].[FK_tbl_pedido_tbl_usuario]', 'F') IS NOT NULL
    ALTER TABLE [dbo].[tbl_pedido] DROP CONSTRAINT [FK_tbl_pedido_tbl_usuario];
GO
IF OBJECT_ID(N'[dbo].[FK_tbl_usuario_relatorio_tbl_relatorio]', 'F') IS NOT NULL
    ALTER TABLE [dbo].[tbl_usuario_relatorio] DROP CONSTRAINT [FK_tbl_usuario_relatorio_tbl_relatorio];
GO
IF OBJECT_ID(N'[dbo].[FK_tbl_usuario_relatorio_tbl_usuario]', 'F') IS NOT NULL
    ALTER TABLE [dbo].[tbl_usuario_relatorio] DROP CONSTRAINT [FK_tbl_usuario_relatorio_tbl_usuario];
GO
-- Dropping existing tables
IF OBJECT_ID(N'[dbo].[tbl_pedido]', 'U') IS NOT NULL
    DROP TABLE [dbo].[tbl_pedido];
GO
IF OBJECT_ID(N'[dbo].[tbl_relatorio]', 'U') IS NOT NULL
    DROP TABLE [dbo].[tbl_relatorio];
GO
IF OBJECT_ID(N'[dbo].[tbl_usuario]', 'U') IS NOT NULL
    DROP TABLE [dbo].[tbl_usuario];
GO
IF OBJECT_ID(N'[dbo].[tbl_usuario_relatorio]', 'U') IS NOT NULL
    DROP TABLE [dbo].[tbl_usuario_relatorio];
GO
CREATE TABLE [dbo].[Pedido] (
    [numero_pedido] varchar(6)  NOT NULL,
    [codigo_cliente] varchar(6)  NULL,
    [condicao_pagamento] varchar(3)  NULL,
    [localizacao] varchar(3)  NULL,
    [descricao_localizacao] varchar(60)  NULL,
    [momento_localizacao] datetime  NULL,
    [id_usuario] int  NULL,
    [data_emissao] datetime  NULL,
    [transportadora] varchar(6)  NULL,
    [representante] varchar(6)  NULL,
    [tipo_frete] varchar(1)  NULL,
    [moeda] decimal(18,0)  NULL,
    [observacao_pedido] varchar(80)  NULL
);
GO
CREATE TABLE [dbo].[Relatorio] (
    [id_relatorio] int IDENTITY(1,1) NOT NULL,
    [id_setor] int  NOT NULL,
    [id_assunto] int  NOT NULL,
    [id_connection_string] int  NOT NULL,
    [nome_interno] varchar(100)  NOT NULL,
    [nome_externo] varchar(150)  NOT NULL,
    [descricao] varchar(250)  NOT NULL,
    [caminho_relatorio] varchar(300)  NOT NULL,
    [tipo_relatorio] varchar(50)  NOT NULL,
    [impressao] varchar(1)  NULL,
    [data_criacao] datetime  NOT NULL,
    [data_atualizacao] datetime  NULL,
    [data_inativacao] datetime  NULL
);
GO
CREATE TABLE [dbo].[Usuario] (
    [id_usuario] int IDENTITY(1,1) NOT NULL,
    [usuario] varchar(20)  NOT NULL,
    [senha] varbinary(50)  NOT NULL,
    [nome] varchar(70)  NOT NULL,
    [nome_resumido] varchar(20)  NULL,
    [data_nascimento] datetime  NULL,
    [status] varchar(4)  NOT NULL,
    [data_criacao] datetime  NOT NULL,
    [data_atualizacao] datetime  NULL,
    [data_inativacao] datetime  NULL
);
GO
CREATE TABLE [dbo].[UsuarioRelatorio] (
    [id_usuario] int  NOT NULL,
    [id_relatorio] int  NOT NULL,
    [data_criacao] datetime  NOT NULL,
    [data_atualizacao] datetime  NULL,
    [data_inativacao] datetime  NULL
);
GO
ALTER TABLE [dbo].[Pedido]
ADD CONSTRAINT [PK_Pedido]
    PRIMARY KEY CLUSTERED ([numero_pedido] ASC);
GO

ALTER TABLE [dbo].[Relatorio]
ADD CONSTRAINT [PK_Relatorio]
    PRIMARY KEY CLUSTERED ([id_relatorio] ASC);
GO

ALTER TABLE [dbo].[Usuario]
ADD CONSTRAINT [PK_Usuario]
    PRIMARY KEY CLUSTERED ([id_usuario] ASC);
GO

ALTER TABLE [dbo].[UsuarioRelatorio]
ADD CONSTRAINT [PK_UsuarioRelatorio]
    PRIMARY KEY CLUSTERED ([id_usuario], [id_relatorio] ASC);
GO

ALTER TABLE [dbo].[UsuarioRelatorio]
ADD CONSTRAINT [FK_tbl_usuario_relatorio_tbl_relatorio]
    FOREIGN KEY ([id_relatorio])
    REFERENCES [dbo].[Relatorio]
        ([id_relatorio])
    ON DELETE NO ACTION ON UPDATE NO ACTION;

CREATE INDEX [IX_FK_tbl_usuario_relatorio_tbl_relatorio]
ON [dbo].[UsuarioRelatorio]
    ([id_relatorio]);
GO

ALTER TABLE [dbo].[UsuarioRelatorio]
ADD CONSTRAINT [FK_tbl_usuario_relatorio_tbl_usuario1]
    FOREIGN KEY ([id_usuario])
    REFERENCES [dbo].[Usuario]
        ([id_usuario])
    ON DELETE NO ACTION ON UPDATE NO ACTION;
GO

ALTER TABLE [dbo].[Pedido]
ADD CONSTRAINT [FK_tbl_pedido_tbl_usuario]
    FOREIGN KEY ([id_usuario])
    REFERENCES [dbo].[Usuario]
        ([id_usuario])
    ON DELETE NO ACTION ON UPDATE NO ACTION;

CREATE INDEX [IX_FK_tbl_pedido_tbl_usuario]
ON [dbo].[Pedido]
    ([id_usuario]);
GO
```

**Figura 17 - *Script* gerado na linguagem de definição de dados do SQL**

Na próxima seção realiza-se uma pequena análise dos *scripts* enviados para a base de dados, o qual pode ser obtido através de recursos do próprio IDE como o *IntelliTrace*, ferramenta disponível em tempo de execução.

## 4.4 ANÁLISE DE *SCRIPTS* GERADOS PELA FERRAMENTA

Analisando o *script* gerado pelo *framework* de persistência para criação da base de dados, Figura 17, nota-se a existência de comandos para prevenir falhas de execução como a criação do *schema* e a exclusão de objetos existentes na base de dados, chaves estrangeiras e tabelas. Durante o ciclo de desenvolvimento, pode ser que algumas relações já contenham registros e por isso, toda vez, é importante verificar se o *script* não irá excluir estas relações. Outro ponto relevante é a ordem dos comandos, onde após prevenir as falhas, primeiro criam-se todas as tabelas e somente depois, as chaves primárias e chaves estrangeiras. Isto é importante para evitar, por exemplo, problemas referentes à inserção de dados em tabelas com relacionamento ou a criação de chaves estrangeiras sem a criação de uma chave primária, o que pode gerar erros de execução do *script*.

Com relação ao comando gerado pelo *Entity Framework*, Figura 18, no carregamento da tela de consulta da entidade Relatorio, a instrução SQL enviada para o banco de dados foi capturada pelo *IntelliTrace* e nota-se uma consulta simples onde por padrão a tabela e as colunas receberam um pseudônimo, também conhecido como *alias* no SQL.

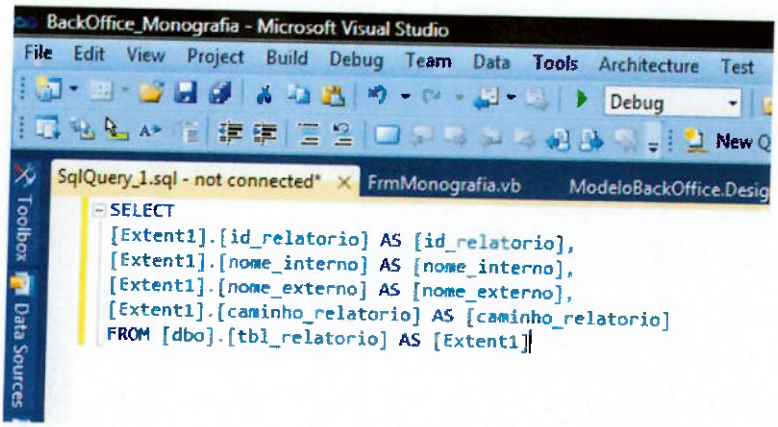


**Figura 18 - Consulta SQL gerada pelo *Entity Framework*.**

Ao executar o método "*SaveChanges*" para persistência de um novo objeto do tipo Relatorio, o *Entity Framework* gera um comando de inserção (*INSERT*) contendo valores nulos para os atributos não preenchidos do objeto "relatório" e outro selecionando o último id_relatorio criado pela numeração automática do banco de dados, segundo Figura 19.

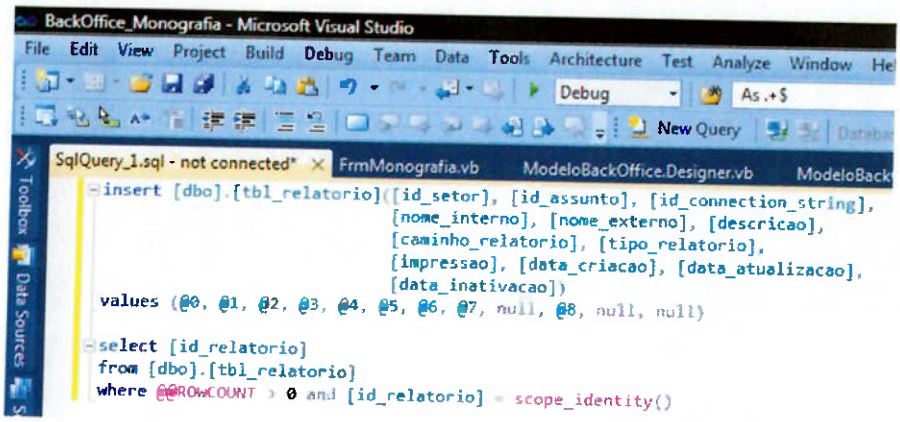


**Figura 19 - Comando SQL para inserção de novo objeto.**

No que se refere à atualização dos objetos, além de inserir um novo objeto, o método "*SaveChanges*" do *Entity Framework* também pode ser utilizado para gravar novos valores nos atributos de um objeto existente ou excluir objetos, gerando por exemplo instruções otimizadas de atualização (*UPDATE*) ou exclusão (*DELETE*) da linguagem SQL.

Com relação à instrução SQL gerada para a pesquisa de relatórios contendo no nome a expressão "TCC" e a descrição preenchida, Figura 20, é importante notar que:

a) O código *String.Empty* gera uma variável de nome @p__linq__0;

b) O código *Contains* gera uma instrução *LIKE* do SQL com "%" dos dois lados.

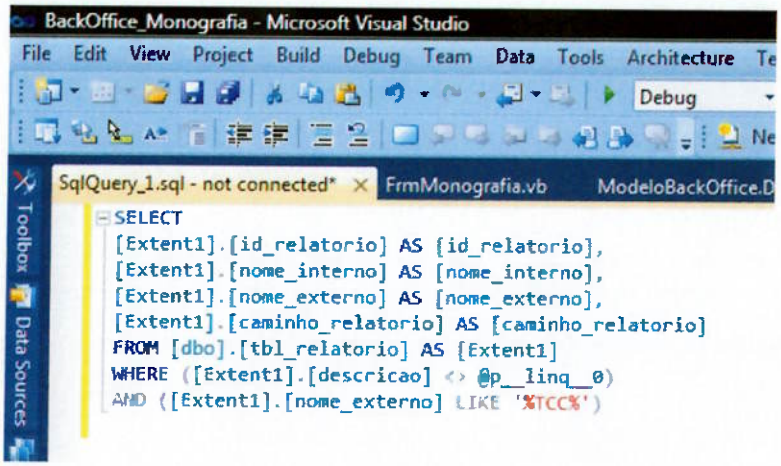


**Figura 20 - Instrução SQL para realizar uma pesquisa com filtro.**

Para o primeiro item, pode-se trocar o código *String.Empty* por simplesmente duas aspas duplas (""), as quais geram uma nova instrução SQL (N'' <> [Extent1].[descricao]) que não se utiliza de variável, melhorando assim o desempenho da consulta. No segundo item, é importante usar o comando *Contains* com prudência, pois a instrução gerada *LIKE* contendo dois "%", dependendo da quantidade de registros, onera muito o desempenho da consulta por ser uma pesquisa textual que, em alguns casos, pode não considerar os índices definidos no modelo relacional. Neste exemplo, fica evidente a necessidade de se ter cuidado ao programar uma consulta com o LINQ e a importância de se conhecer os comandos gerados pela ferramenta de persistência.

Na próxima seção demonstra-se uma proposta contendo a estratégia para implantação do *Entity Framework* na parte escolhida da aplicação *BackOffice* com diversas classes existentes e um banco de dados compartilhado com o sistema ERP.

## 4.5 PROPOSTA DE IMPLANTAÇÃO DO *FRAMEWORK* DE PERSISTÊNCIA

A proposta a seguir corresponde a um guia resumido das etapas que os profissionais de TI da empresa Gratex precisam seguir para atualizar o sistema *BackOffice* e utilizar, se possível, um *framework* de persistência como o *Entity Framework* do *Visual Studio* 2010. O objetivo desta estratégia é a redução do acoplamento entre as camadas da aplicação, o aumento da produtividade nas manutenções e a flexibilidade de se trabalhar com SGBDs diferentes.

Em um primeiro momento, os Analistas Programadores e o DBA da empresa podem participar de treinamentos e cursos sobre modelos relacionais, normalização, modelos de objetos, conceitos de UML e padrões de projeto para aumentar ou nivelar o conhecimento técnico entre as pessoas envolvidas neste projeto de implantação.

A partir do entendimento e construção de uma visão ampla sobre o modelo objeto e o modelo relacional, da mesma forma que realizada nesta pesquisa, abre-se caminho para o estudo do mapeamento e do *framework* de persistência. Neste momento, é interessante que todos os profissionais de TI visualizem o problema de descasamento de impedância entre orientação a objetos e SGBD relacional.

Após a conclusão do estudo, por exemplo, do *Entity Framework*, analisando suas vantagens e desvantagens, seus recursos e limitações, o Analista Programador precisa realizar uma melhoria significativa nos diagramas de classe, assim como descrito na seção 3.2.1.

Com o modelo de objetos seguindo os conceitos da UML em conjunto com os padrões de projeto e o modelo relacional normalizado, no mínimo até a 2FN, pode-se dizer que o ambiente está preparado para o uso inicial do *Entity Framework*.

Da mesma forma que o realizado na seção B.1 do apêndice B, pode-se gerar o EDM a partir de um modelo relacional existente, neste caso, o banco de dados compartilhado com o sistema ERP da empresa.

Com o EDM disponível para uso, o código fonte da aplicação relacionada a salvar ou modificar um objeto, pode ser trocado aos poucos, conforme a solicitação por manutenções nas telas existentes, evitando um trabalho inicial muito exaustivo. No caso de telas novas, o desenvolvimento pode utilizar apenas o *Entity Framework*

para acesso ou modificação dos dados, sem a necessidade de submeter às modificações diretamente ao SGBD.

Até este momento a solução descrita não apresenta maiores complicações, no entanto, quando alguma alteração for realizada no modelo relacional pelo sistema ERP, o qual não se utiliza do mesmo *framework* de persistência, ocorrerá uma divergência entre o modelo vinculado ao *Entity Framework* do *BackOffice* e o modelo relacional existente na base de dados.

Para resolver este impasse, a solução depende de um processo humano envolvendo o DBA e os Analistas Programadores de ambas as equipes (.NET e ADVPL) para sincronizar este tipo de modificação no modelo relacional com uma simples atualização do EDM através da opção atualizar o modelo segundo uma base de dados existente, demonstrada na seção B.2 do apêndice B.

Com esse processo estabelecido entre os profissionais de TI, não existirão as possíveis divergências entre o EDM e o modelo relacional, as quais poderiam inviabilizar esta proposta de implantação.

Na prática, o sucesso desta proposta ainda pode depender de um estudo mais aprofundado de outras partes do sistema, não contempladas nesta pesquisa, e do empenho dos profissionais que desenvolvem a aplicação *BackOffice*, os quais podem ser motivados por um líder ou aprendizado pessoal ou ainda, por exemplo, a partir da necessidade de troca do SGBD do sistema ERP.

A utilização desta proposta compreende apenas ao passo inicial para a implantação do *framework* de persistência, porém o trabalho de adequação não é simples e exigirá um esforço considerável dos profissionais de TI.

# 5 COMENTÁRIOS FINAIS E CONCLUSÃO

O problema da dissonância entre bancos de dados relacionais e o modelo orientado a objetos é constante em projetos de *softwares*. A partir do estudo realizado entende-se que, além da escolha de um *framework* de persistência, o arquiteto de sistema precisa propor uma adequação da modelagem relacional e do diagrama de classes UML no caso de sistemas legados, para se ter um resultado apropriado com o uso da ferramenta de mapeamento.

Mesmo com uma abordagem não muito profunda das funcionalidades do *Entity Framework*, pode-se concluir que esta ferramenta é uma boa opção quando for necessário trabalhar com um modelo de objetos persistindo em um SGBD relacional.

No desenvolvimento de um sistema novo, recomenda-se o uso de um *framework* de persistência, por exemplo, o *Entity Framework*, para evitar trabalhos manuais de mapeamento entre os modelos de objetos para relacional e melhorar a produtividade do programador na elaboração do sistema que atenda as regras de negócio da empresa.

No desenvolvimento de novas funcionalidades em sistemas legados é importante verificar a possibilidade de utilizar um modelo híbrido, com as partes novas utilizando o *framework* de persistência para a geração automática da base de dados e as partes já existentes mantendo seu mapeamento manual. Em outras palavras, mantendo a base de dados já existente e usando o *framework* para vincular classes às tabelas e permitir o acesso a estas últimas a partir do *framework*. Através desta pesquisa, entende-se que esse modelo de solução está relacionado diretamente à complexidade do sistema, quanto mais complexo, mais difícil será a introdução da ferramenta para o mapeamento. Em alguns casos, como o *BackOffice* da empresa Gratex, a utilização pode ser adequada somente após a adoção dos padrões de projeto e normalização em todo o sistema existente.

Após a aplicação prática, constata-se que a forma de escrever um determinado código-fonte altera com relevância o *script* gerado pelo *framework* de persistência, *Entity Framework*, no que se refere aos comandos para manipulação dos dados no SGBD, resultando em desempenho melhor ou pior se comparados

entre si. Assim, torna-se importante a definição de políticas sobre o uso do *framework* de persistência e treinamento específico da equipe de desenvolvedores da Gratex, para maximizar os benefícios com a aplicação da ferramenta e minimizar os problemas relacionados a diferentes códigos-fonte que executam a mesma tarefa e não possuem desempenho semelhante.

Com esse estudo, um questionamento a ser feito em trabalhos futuros é se os profissionais de informática estão dispostos a mudarem o seu mecanismo de trabalho após a introdução de um *framework* de persistência. Por exemplo, um Analista Programador, ao usar a ferramenta de mapeamento exigirá um nível maior de acesso para alterações no banco de dados, o qual é de responsabilidade do Administrador de Banco de Dados.

Uma possível continuação desse trabalho é o estudo de sistemas gerenciadores de banco de dados como o Caché, produzido pela *InterSystems*, que através de uma arquitetura unificada de dados propõe a eliminação da incompatibilidade entre objeto e visões relacionais dos dados e o uso de um *framework* de persistência, porém mantendo a possibilidade de trabalhar em conjunto com o modelo relacional. Neste caso, a integração do SGBD com as aplicações orientadas a objeto não seria dissonante e possivelmente existiria um melhor desempenho ao armazenar objetos por não precisar decompô-los em linhas de tabelas (CACHÉ, 2011).

# REFERÊNCIAS

**CACHÉ. 2011.** Informações sobre Caché. *InterSystems Brasil.* [Online] InterSystems, 2011. [Citado em: 26 de Janeiro de 2011.] http://www.intersystems.com.br/conteudos/pgpadrao.asp?MTU6NTI6MDV8MjY4.

**DATE, C. J. 2005.** *Introdução a Sistemas de Banco de Dados.* s.l. : Campus, 2005.

**ELMASRI, Ramez e NAVATHE, Shamkrant. 2010.** *Fundamentals of Database Systems.* 6a. edição. s.l. : Addison-Wesley, 2010.

**FOWLER, Martin. 2003.** *Padrões de Arquitetura de Aplicações Corporativas.* São Paulo : Bookman, 2003. pp. 59-65.

**—. 2005.** Patterns in Enterprise Software. *martin Fowler .com.* [Online] 2005. [Citado em: 12 de Setembro de 2010.] http://martinfowler.com/articles/enterprisePatterns.html.

**GAMMA, Erich, et al. 1994.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Pearson Education, 1994.

**JACOBSON, Ivar; BOOCH, Grady and RUMBAUGH, James. 1999.** *The Unified Software Development Process.* s.l. : Addison-Wesley Professional, 1999. 0201571692.

**JOHNSON, Rod. 2002.** *Expert One-on-One J2EE Design and Development.* s.l. : Wrox Press, 2002. 1861007841.

**LARMAN, Craig. 2004.** *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* 3ª. s.l. : Prentice Hall, 2004.

**MACORATTI, José Carlos. 2009.** Desvendando o Entity Framework - O Entity Data Model (EDM). *iMasters.* [Online] Abril, 15 de Julho de 2009. [Citado em: 18 de Janeiro de 2011.]

http://imasters.com.br/artigo/13549/desenvolvimento/desvendando_o_entity_framework_o_entity_data_model_edm/.

**MEHTA, Vijay P. 2008.** *Pro LINQ Object Relational Mapping with C# 2008: Discover the power that LINQ to SQL and LINQ to Entities can bring to your projects.* New York : APRESS, 2008.

**MELO, Ana Cristina. 2004.** *Desenvolvendo Aplicações com UML 2.0.* 2. s.l. : Brasport, 2004. pp. 17-22;93-130. 8574521752.

**MORAES, Claudio Bernardo Guimarães de. 2010.** Porte da empresa. *BNDES - O banco nacional do desenvolvimento.* [Online] BNDES, 05 de Março de 2010. [Citado em: 28 de Outubro de 2010.] http://www.bndes.gov.br/SiteBNDES/bndes/bndes_pt/Navegacao_Suplementar/Perfil/porte.html.

**MSDN. 2011.** Expressões lambda (guia de programação de C#). *MSDN Comunidade de Desenvolvedores.* [Online] Microsoft, 2011. [Citado em: 25 de Janeiro de 2011.] http://msdn.microsoft.com/pt-br/library/bb397687.aspx.

**MULLER, Robert J. 2002.** *Projeto de Banco de Dados – Usando UML para modelagem de dados.* s.l. : Berkeley, 2002.

**NASCIMENTO, Natália de Lima do. 2009.** Persistência em Banco de Dados: Um Estudo Prático. *Centro de Informática da UFPE.* [Online] Novembro de 2009. [Citado em: 07 de Novembro de 2010.] http://www.cin.ufpe.br/~tg/2009-2/nln.pdf.

**NHIBERNATE, Community. 2010.** NHibernate - Relational Persistence for Idiomatic .NET. *NHibernate Forge.* [Online] 2010. [Citado em: 11 de Dezembro de 2010.] http://nhforge.org/doc/nh/en/index.html.

**PAULI, Guinther. 2010.** Classes POCO: ADO.NET Entity Framework com .NET 4. *DevMedia.* [Online] 2010. [Citado em: 07 de Novembro de 2010.] http://www.devmedia.com.br/post-17003-ADO-NET-Entity-Framework-com--NET-Framework-4--Classes-POCO-Novidades-no-Visual-Studio-2010-Parte-20.html.

**PINHEIRO, José Francisco Viana. 2005.** Um framework para persistência de Objetos em Banco de Dados Relacionais. *SISTEMA DE PUBLICAÇÃO ELETRÔNICA DE TESES E DISSERTAÇÕES.* [Online] 2005. [Citado em: 01 de Setembro de 2010.] http://www.bdtd.ndc.uff.br/tde_busca/arquivo.php?codArquivo=2397.

**ROMAN, Ed; AMBLER, Scott W. e JEWELL, Tyler. 2002.** *Dominando Enterprise JavaBeans.* São Paulo : Bookman, 2002. p. 106.

**SILBERCHATZ, A.; KORTH, H. e SUDARSHAN, S. 2006.** *Sistemas de Banco de Dados.* 1ª. s.l. : Campus, 2006.

**W3C. 2008.** Extensible Markup Language (XML) 1.0 (Fifth Edition). *W3C.* [Online] World Wide Web Consortium, 26 de Novembro de 2008. [Citado em: 27 de Janeiro de 2011.] http://www.w3.org/TR/xml/.

# APÊNDICE A - PADRÕES DE PROJETO

Para o entendimento dos padrões de projeto é necessário conhecimento da modelagem relacional e UML como Diagramas de Classes, Objetos e Seqüência.

## A.1 TIPOS DE PADRÕES DE PROJETO

Os padrões de projeto listados a seguir são classificados em (FOWLER, 2005):

a) Estrutural - orienta a construção e organização das classes;
b) Comportamental - trata as interações e divisões de responsabilidades entre as classes;
c) Metadados - possibilita a criação de códigos genéricos do mapeamento.

### Estrutural

**Campo Identificador:**

Diferente do Banco de Dados relacional, o objeto trabalha com serialização e não tem o conceito de chave primária, ou seja, um atributo ou combinação de atributos que possuem a propriedade de identificar de forma única uma linha da tabela para evitar a duplicidade e recuperar somente a linha correspondente à chave. Para solucionar esse impasse, defini-se um atributo na classe representando a chave primária, conforme mostrado na Figura 21.

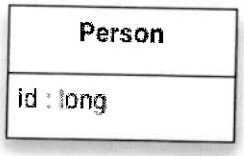


**Figura 21 - Identificação de objetos através do atributo *id*.**
**Fonte: (FOWLER, 2005).**

**Mapeamento de chave estrangeira:**

No modelo relacional quando existe um relacionamento simples entre duas tabelas, muitos-para-um ou um-para-um, se utiliza o campo de chave estrangeira, ou seja, uma tabela exporta sua chave primária para a outra tabela. Como no modelo orientado a objetos, o relacionamento ocorre a partir de uma referência, esse padrão de projeto propõe mapear essa referência para a chave estrangeira respectiva no Banco de Dados. Segue exemplo na Figura 22.

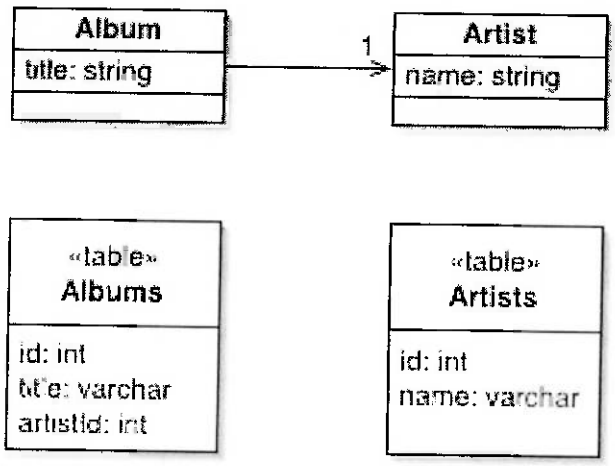


**Figura 22 - Relacionamento entre objetos através do campo *artistId*.**
**Fonte: (FOWLER, 2005).**

**Mapeamento de tabela associativa:**

Os objetos, nativamente, manipulam atributos multivalorados, preenchendo-os com coleções de objetos de outra classe, porém o Banco de Dados relacional não compartilha dessa capacidade e trabalha apenas com um valor por campo. Quando ocorre um relacionamento de muitos-para-muitos, esse padrão de projeto traz uma solução clássica através da criação de uma tabela intermediária com chaves estrangeiras para as tabelas que estejam vinculadas pela associação, conforme visualizado na Figura 23.

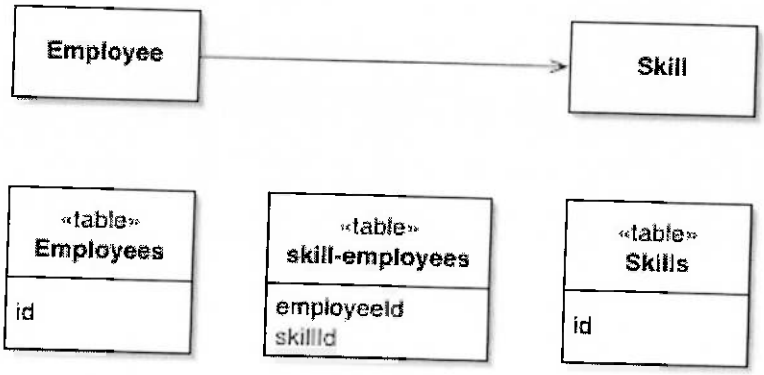


**Figura 23 - Associação de objetos através da tabela *skill-employees*.**
**Fonte: (FOWLER, 2005).**

### Atributo composto:

Na modelagem de Banco de Dados relacional não se representam atributos compostos em apenas um campo da tabela. Neste caso, utilizam-se dois campos na tabela correspondente ao objeto. Por exemplo, na Figura 24 o atributo *period* do tipo *DateRange* contém um intervalo, o qual é mapeado na tabela Employments em dois campos, start e end, contendo respectivamente os valores do limite inferior e do limite superior do intervalo.

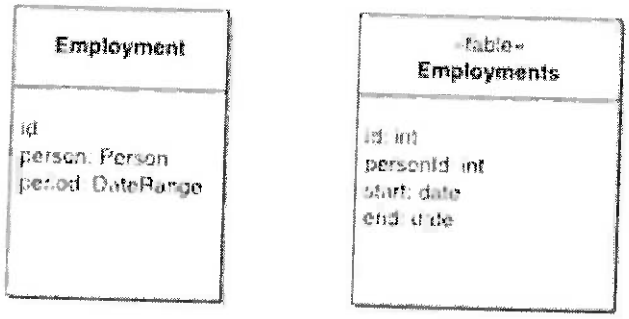


**Figura 24 - Atributo composto, *period*, dividido em dois campos, *start* e *end*.**
**Fonte: (FOWLER, 2005).**

### Herança em Tabela Única:

Diferente da orientação a objetos, no modelo relacional não existe o conceito de herança. Dentre os diversos modos de representar a herança na modelagem de dados relacional, um dos mais simples é o padrão de projeto Herança em Tabela Única, o qual representa a classe pai e todas as suas filhas na mesma tabela, ver Figura 25.

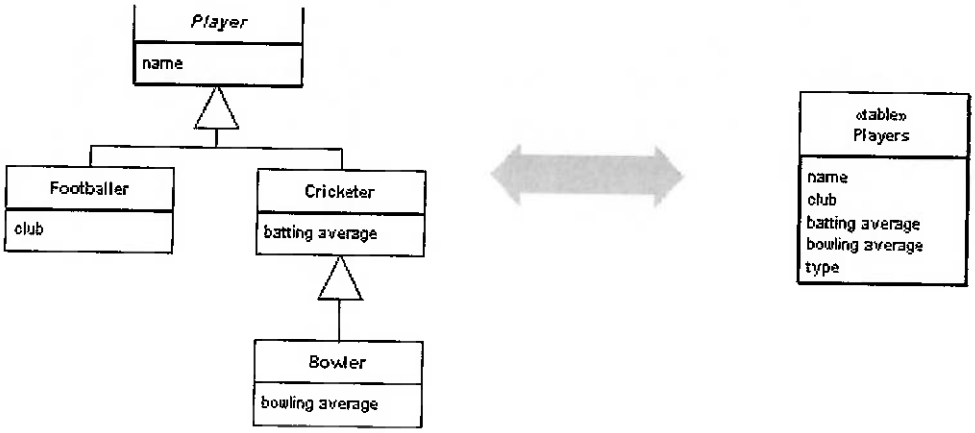


**Figura 25 - Herança no modelo relacional com uma tabela para todas as classes.**
**Fonte: (FOWLER, 2005).**

### Herança com Tabela por Classe:

Outro modo de fazer a representação de uma hierarquia de herança de classes é utilizar uma tabela por classe, ver Figura 26.

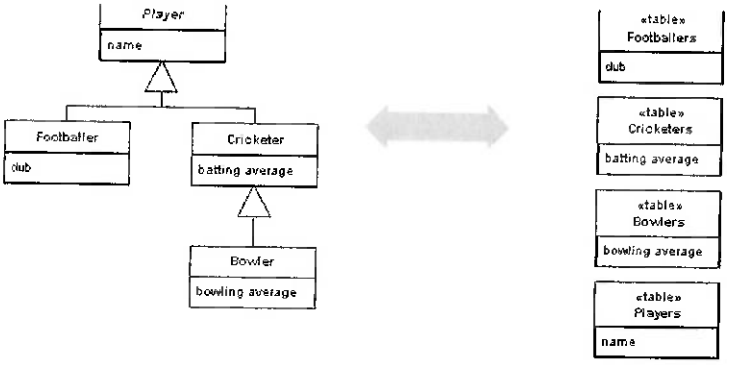


**Figura 26 - Herança no modelo relacional com uma tabela por classe.**
**Fonte: (FOWLER, 2005).**

### Herança com Tabela por Classe Concreta:

Para mapear a representação de uma hierarquia de herança de classes, outra solução é criar uma tabela por classe concreta, ver Figura 27.

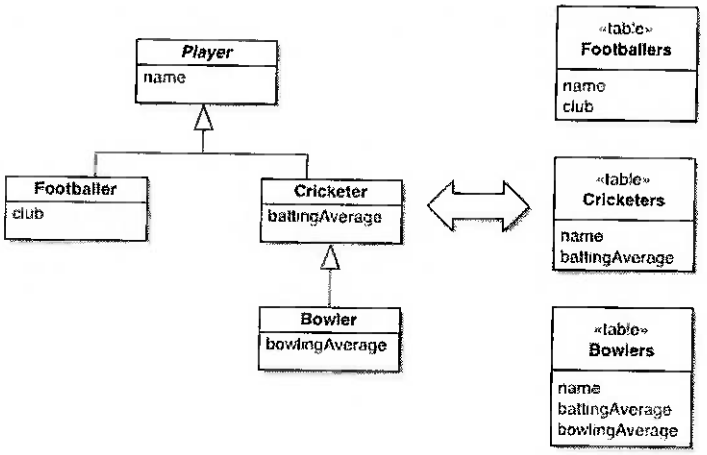


**Figura 27 - Herança no modelo relacional com uma tabela por classe concreta.**
**Fonte: (FOWLER, 2005).**

## Comportamental

### Unidade de Trabalho:

O padrão de projeto comportamental Unidade de Trabalho mantém uma lista de objetos afetados por uma transação de negócio e coordena a gravação das alterações e a resolução de problemas de concorrência. Com esse mecanismo, o número de chamadas ao Banco de Dados é reduzido e evita-se a leitura de dados inconsistentes. Segue na Figura 28, uma classe modelo com métodos desse padrão.

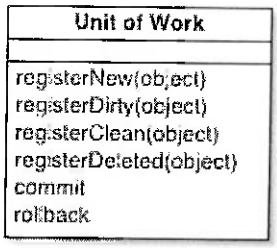


**Figura 28 - Objetos afetados por uma transação de negócio**
**Fonte: (FOWLER, 2005).**

**Mapa de Identificador:**

Com a utilização desse padrão de projeto, cada objeto é carregado apenas uma vez mantendo, objetos carregados em um mapa. Quando existe referência para esses objetos, a busca utiliza o mapa primeiro, evitando acessos ao Banco de Dados, conforme mostrado na Figura 29.

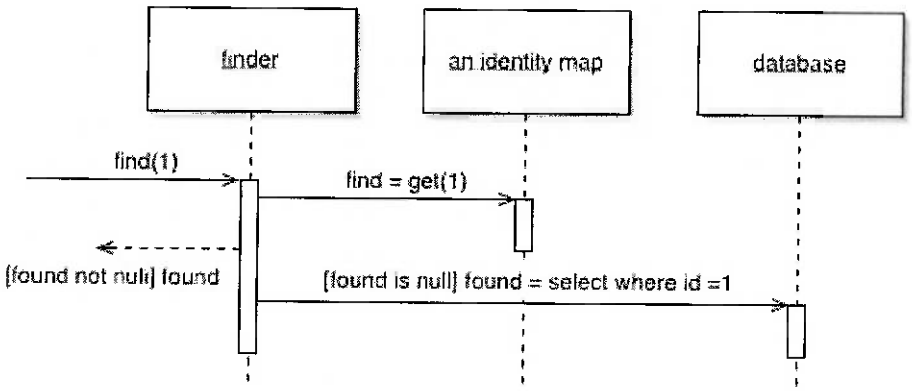


**Figura 29 - Mapeamento dos objetos carregados na memória.**
**Fonte: (FOWLER, 2005).**

**Carregamento Tardio:**

O padrão de projeto Carregamento Tardio refere-se a quando um objeto não contém todos os dados necessários, mas conhece como obtê-los, como visualizado na Figura 30.

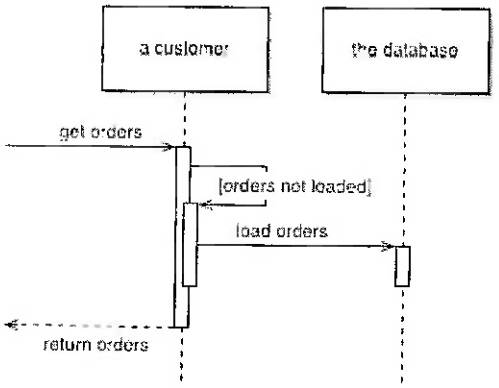


**Figura 30 - Obtenção de dados que o objeto não possui.**
**Fonte: (FOWLER, 2005).**

## Metadados

### Mapeamento de metadados:

Para habilitar o desenvolvimento de códigos genéricos do mapeamento objeto-relacional, se faz necessário o uso desse padrão, pois com os detalhes do mapeamento objeto-relacional armazenados em metadados é possível identificar qual campo da tabela contém o dado de um atributo do objeto em memória, conforme mostrado na Figura 31.

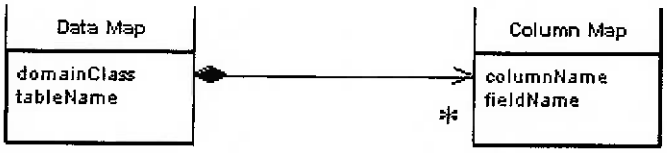


**Figura 31 - Mapeamento objeto-relacional a partir de metadados.**
**Fonte: (FOWLER, 2005).**

### Objeto Consulta:

No padrão de projeto Objeto Consulta, existe um objeto para representar uma consulta ao Banco de Dados, buscando tornar o código independente do Esquema de Banco de Dados, ao possibilitar a construção de consultas referenciando classes e atributos ao invés de tabelas e campos. Segue exemplo na Figura 32.

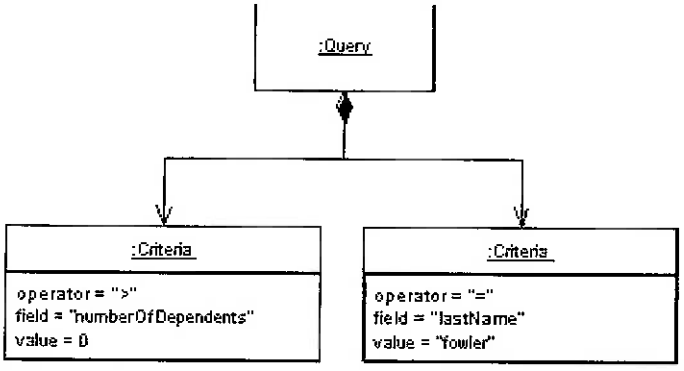


**Figura 32 - Representação da consulta ao Banco de Dados em objeto.**
**Fonte: (FOWLER, 2005).**

**Repositório:**

A criação da camada de aplicação mediadora entre as camadas de domínio e de mapeamento de dados, composta por uma interface do tipo coleção para acessar os objetos do domínio, corresponde ao padrão de projeto Repositório, observado na Figura 33.

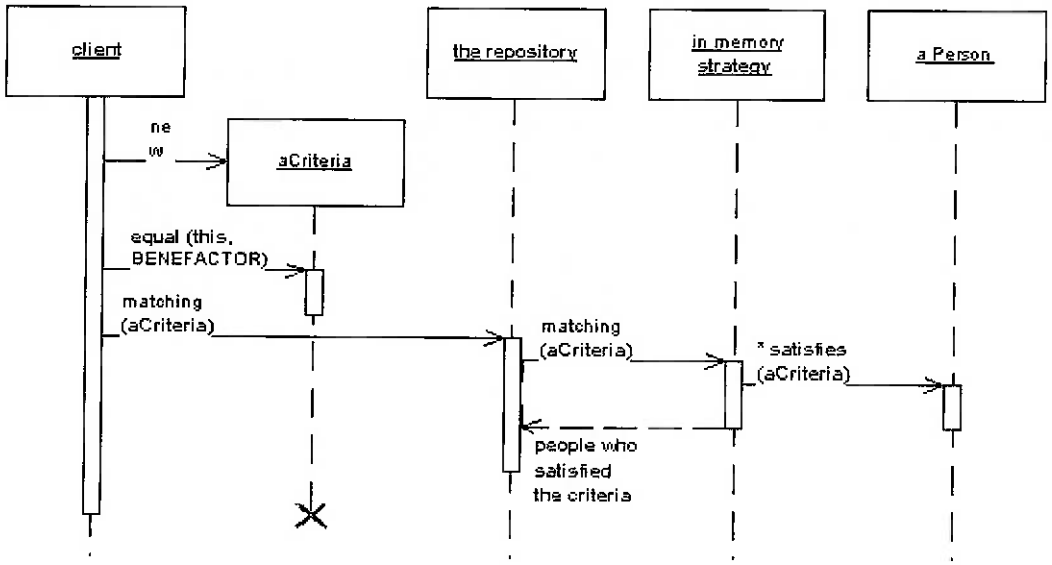


**Figura 33 - Repositório como gerenciador de coleções de objetos.**
**Fonte: (FOWLER, 2005).**

A partir dos padrões de projeto listados, facilita-se o entendimento dos *frameworks* de persistência que automatizam o mapeamento objeto-relacional, tanto no funcionamento quanto nas suas características.

# APÊNDICE B - *ENTITY FRAMEWORK*

A seguir é demonstrado, de modo prático, procedimentos para utilização do *Entity Framework* no IDE *Visual Studio* 2010.

## B.1 GERAÇÃO DO *ENTITY DATA MODEL*

O *Entity Data Model* refere-se ao modelo representativo do banco de dados a ser gerado ou existente como no caso escolhido. Ao abrir um projeto no *Visual Studio* 2010, adicione um novo item e escolha a opção ADO .NET *Entity Data Model*, conforme visualizado na Figura 34.

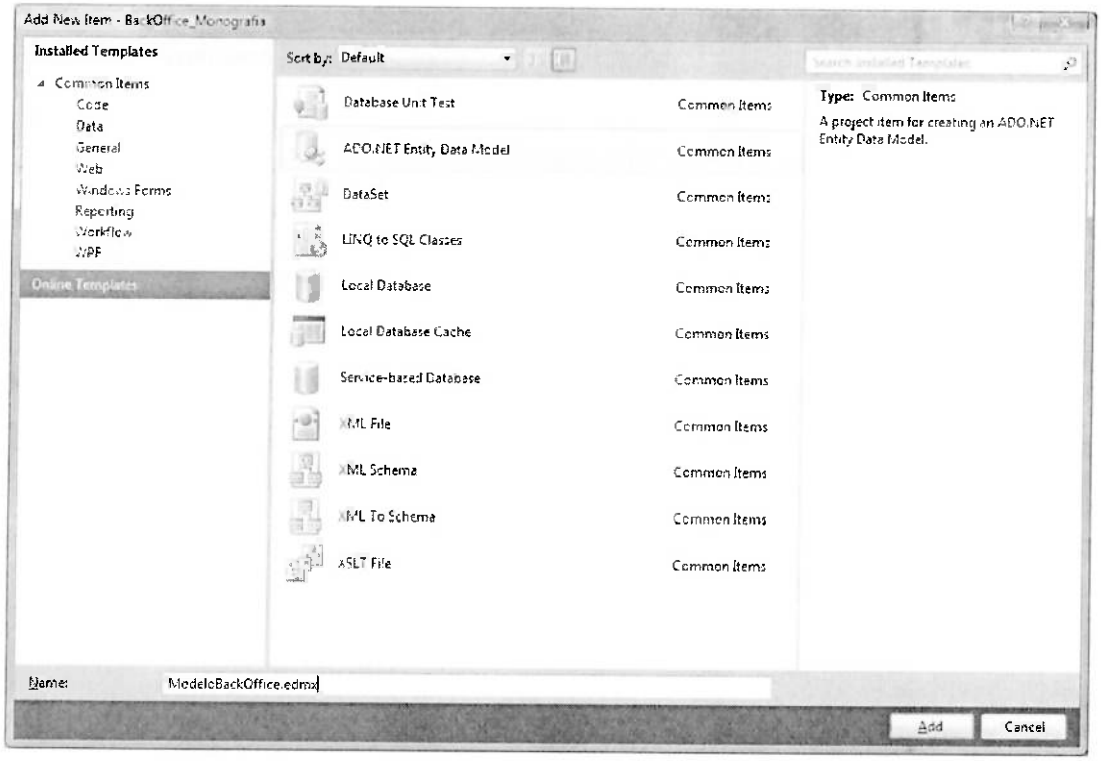


**Figura 34 - Itens para adicionar no projeto do *Visual Studio*.**

Após definir um nome para o arquivo do modelo, escolha a opção "*Generate from database*", indicada na Figura 35, e configure uma conexão para o SGBD e base de dados a ser mapeada.


**Figura 35 - Definição da origem do conteúdo de um *Entity Data Model*.**

Com o acesso configurado, selecione os objetos do banco de dados, tabelas, *stored procedures* e *views*, que serão mapeados e vinculados ao *Entity Data Model*, ver Figura 36.

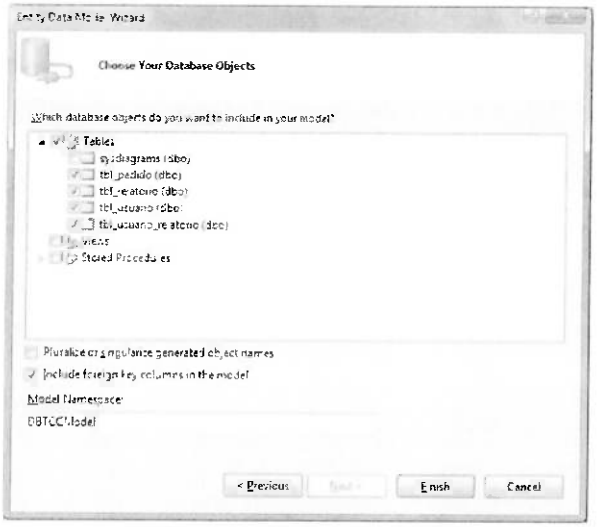


**Figura 36 - Escolha dos objetos de banco de dados listados no *Entity Data Model*.**

Ao clicar no botão "*Finish*" da Figura 36, o modelo representativo é gerado mediante ao esquema do banco de dados, criando os relacionamentos das entidades, listando os atributos e estabelecendo os fluxos de navegação entre as entidades conforme Figura 11 da seção 4.1.

## B.2 CRIAÇÃO DA BASE DE DADOS A PARTIR DO EDM

Com a solução do *Visual Studio* 2010 aberta, selecione a opção "*Model Browser*", conforme Figura 37, e escolha entre gerar uma nova base de dados a partir do modelo ou atualizar o modelo segundo uma base de dados existente. Neste caso, selecione a segunda opção para gerar o *script* na linguagem de definição de dados do SQL.

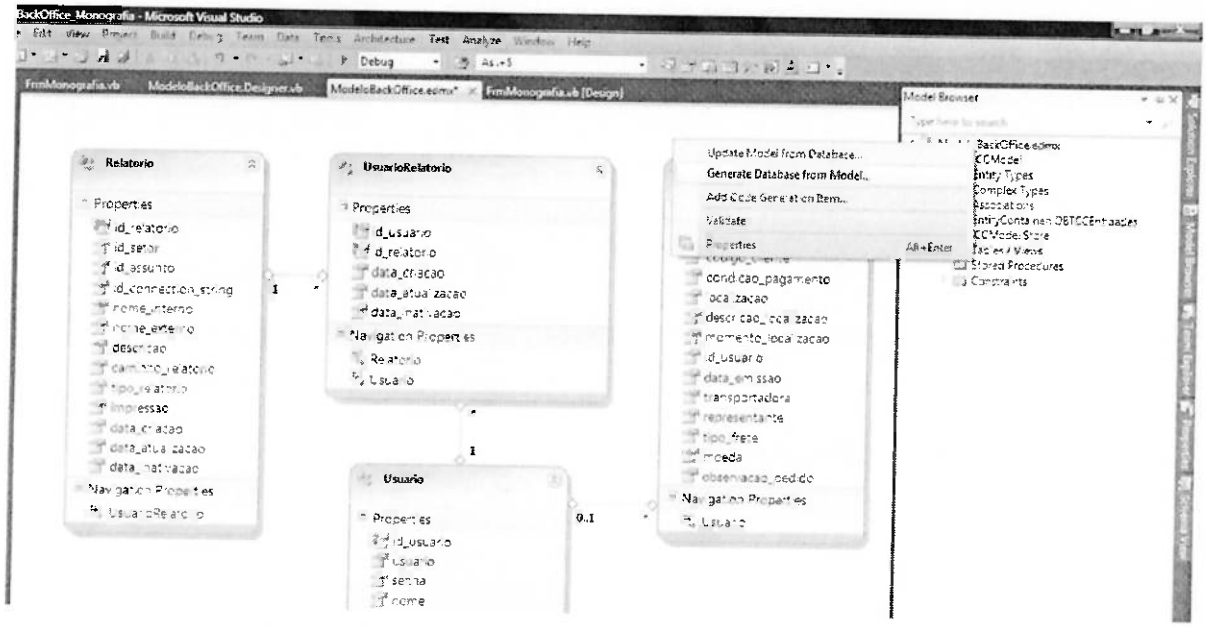


**Figura 37 - Criação da base de dados a partir do EDM.**

O *script* gerado pela ferramenta de persistência encontra-se na seção 4.3 e a sua análise está localizada na seção 4.4.